ENTERPRISE ANALYSIS OF STRATEGIC AIRLIFT TO OBTAIN COMPETITIVE ADVANTAGE THROUGH FUEL EFFICIENCY

DISSERTATION

Adam D. Reiman, Lt Col, USAF

AFIT-ENS-DS-14-S-16

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-DS-14-S-16

ENTERPRISE ANALYSIS OF STRATEGIC AIRLIFT TO OBTAIN COMPETITIVE ADVANTAGE THROUGH FUEL EFFICIENCY

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Adam D. Reiman, BS, MS

Lt Col, USAF

September 2014

AFIT-ENS-DS-14-S-16

ENTERPRISE ANALYSIS OF STRATEGIC AIRLIFT TO OBTAIN COMPETITIVE ADVANTAGE THROUGH FUEL EFFICIENCY

Adam D. Reiman, BS, MS

Lt Col, USAF

Approved:

//signed//	17 Jul 2014
Jeffery D. Weir, PhD (Chairman)	Date
//signed//	17 Jul 2014
Alan W. Johnson, PhD (Member)	Date
//signed//	17 Jul 2014
Maj Thomas E. Dube (Member)	Date

Accepted:

ADEDEJI B. BADIRU, PhD Dean, Graduate School of Engineering and Management	Date

Abstract

The rising cost of fuel has led to increasing emphasis on fuel efficiency in the aviation industry. As fuel costs become a larger proportion of total costs, those entities with a dynamic capability to increase their fuel efficiency will obtain competitive advantage. Fuel efficiency must be assessed simultaneously with cargo throughput which is the primary goal of airlift effectiveness. Assessing cargo throughput and fuel efficiency requires the creation of all routes of potential value for a given set of requirements that need to be airlifted from source to destination airfield. Routing in this context refers to the set of potential sorties from source to destination. This set of potential routings rapidly increases as source and destination approach antipodal points on the globe. The time required for route computation can be significantly reduced through the use of nodal reduction. Computation time is a critical component to the effective operational comparison of routing alternatives based on cargo throughput and fuel efficiency. Use of the proposed model can assist evaluation of enterprise wide efficiency and effectiveness.

To my loving family.
Thank you for all of your support.

Acknowledgements

I would like express sincere appreciation to my dissertation advisor, Dr. Weir, and committee members, Dr. Johnson and Dr. Dube, for their support in the development of this research.

Adam D. Reiman

Table of Contents

List of Figures

Figure Page

List of Tables

List of Abbreviations

ACN	Aircraft Classification Number
AMC	Air Mobility Command
APOD	Aerial Port of Debarkation
APOE	Aerial Port of Embarkation
CAMPS	Consolidated Air Mobility Planning System
CFIT	Controlled Flight Into Terrain
CFL	Critical Field Length
CRAF	Civil Reserve Air Fleet
DAFIF	Digital Aeronautical Flight Information File
FDP	Flight Duty Period
FEI	Fuel Efficiency Index
GCD	Great Circle Distance
GW	Gross Weight
ICAO	International Civil Aviation Organization
JP	Joint Publication
KLBS	Thousand Pounds
MC	Mission Capable
MDS	Mission Design Series
MOG	Maximum on the Ground (Working or Parking)
MTM/D	Million Ton Miles Per Day
NM	Nautical Mile
NPS	Naval Postgraduate School
NRMO	NPS/RAND Mobility Optimizer
OCS	Obstacle Clearance Surface
OD	Origin Destination
PCN	Pavement Classification Number
PFEE	Payload Fuel Energy Efficiency
PMAX	Payload Maximum
RAM	Random Access Memory
RCR	Runway Condition Reading
RSC	Runway Surface Condition
SAP	Strategic Airlift Problem
SDVF	Single Dimension Value Function
SFC	Specific Fuel Consumption
TRANSCAP	Transportation System Capability
TSP	Travelling Salesman Problem
VFT	Value Focused Thinking
VRP	Vehicle Routing Problem

ENTERPRISE ANALYSIS OF STRATEGIC AIRLIFT TO OBTAIN COMPETITIVE ADVANTAGE THROUGH FUEL EFFICIENCY

I. Introduction

Fuel costs are an increasing portion of the total costs for the aviation industry. These rising costs elevate the relative importance of fuel efficiency. Firms developing a dynamic capability to enhance fuel efficiency can obtain a sustained competitive advantage over their competitors. To assist management in developing this dynamic capability, a framework is proposed. Within this framework, several metrics are introduced to assess the success of a firm's fuel efficiency efforts. The one metric that addresses fuel efficiency across the enterprise is the Fuel Efficiency Index (FEI). The FEI was examined against great circle distance, load factor and fuel consumed from a month of Air Mobility Command (AMC) sorties to obtain an understanding of the primary interrelationships.

An airlift enterprise is established to move customer requirements from a source airfield to a destination airfield. With each source and destination airfield, a network of nodes exists representing potential airfields and edges representing potential sorties. The edges leading from source to destination represents a route alternative. The set of all route alternatives can be contrasted using the FEI. Every sortie forming an edge has a value associated with it that is dependent upon the length of the edge. To establish the value associated with sortie distance, a model was created. Airlift planners can use this aircraft specific distance model to eliminate poor alternatives. The model can also be applied to select potential airfields as intermediate fueling or cargo trans-load locations to enhance airlift enterprise effectiveness and efficiency.

The number of airlift nodes represented by International Civil Aviation Organization (ICAO) codes is large enough that executing comparisons against the set of all potential

alternatives is computationally extensive. To enhance the speed of execution for the comparison of alternatives, a set of nodal reduction techniques was developed. The primary technique is based upon the planning distance value model. The decision maker can adjust the weights assigned to the planning distance value model based upon their preferences. Other nodal reduction techniques include the total distance multiple, minimum/effective runway length, minimum runway width, runway pavement strength, departure obstacles and diplomatic clearances. The decision maker can select the values for nodal elimination. This affords the decision maker the opportunity to trade increased execution speed for a decrease in the number of potential alternatives. Since the speed of execution of the route algorithm is dependent upon the pair of airfields selected, airfield pairs were selected at random to obtain a representative example for further analysis.

II. Literature Review

There exists an abundant body of knowledge associated with the theme of enterprise airlift planning. To better understand previous contributions, the literature is broken down into the following categories: airlift metrics, competitive advantage, alliancing, organizational culture, routing, tabu search, value focused thinking and scheduling. A summary of the literature currently referenced in the articles of chapter IV of this dissertation can be seen in Figure 1 below.

Currently Referenced in Articles	Airlift Metrics	Competitive Advantage	Alliancing	Organizational Culture	Routing	Tabu Search	Value Focused Thinking	Scheduling
Ai and Kachitvichyanukul (2009)					●			
Alexander and Hall (1991)	●							
Babikian et al (2001)	●							
Baker et al (2002)	●				●			●
Balakrishnan et al (1989)					●			
Barney (1986, 1991)		●		●				
Bell and McMullen (2004)					●			
Bodin (1990)					●			●
Chang (2008)					●			
Crino et al (2004)					●	●		
Crum and Morrow (2002)								●
Dijkstra (1959)					●			
Dyer and Sarin (1979)							●	
Eisenhardt and Martin (2000)		●						
Gagnepain and Marin (2007)			●					
Gendreau and Soriano (1998)	●							
Hatch (1993)				●				
Held and Karp (1962)					●			
Hileman et al (2008)	●							
Jackson et al (1996, 1999)							●	
Karmarkar (1984)					●			
Keeney (1994)							●	
Lahiri (2003)	●							
Lambert (2007)					●	●		
Lee (2004)	●							
Lei et al (2011)					●			

	Airlift Metrics	Competitive Advantage	Alliancing	Organizational Culture	Routing	Tabu Search	Value Focused Thinking	Scheduling
Lewis (1998)	●							
Li at al (2010)					●			
Longo et al (2006)					●			
Lund (1993)	●							
Lurdes et al (1990)	●							
Martin and Voltes-Dorta (2011)	●							
Mazraati (2010)	●							
Mihram and Nolan (1969)	●				●			
Miravite and Schlegel (2006)							●	
Murphy et al (1989)								●
Nagata et al (2010)					●			
Naylor (2009)	●				●			
Oster et al (2013)	●							
Owen (2008)	●							
Pascal (1665)					●			
Pisinger and Ropke (2007)					●			
Rappaport et al (1992)					●			
Reiman et al (2011)	●							
Rutherford and Zeinali (2009)	●							
Samm and Perelli (1982)								●
Schein (1984)				●				
Schmenner (1998, 2001, 2004)	●							
Sere (2005)	●				●			
Thomchick (1993)	●							
Vincenty (1975)	●							
Watson (2003)	●							
Yamani (1990)	●				●			

Figure 1: Literature Review (Referenced in Articles)

Figure 1 only addresses the body of literature currently referenced in the included articles. There are a number of other sources that add to the body of knowledge that have not been referenced inside of the articles. These sources are included in Figure 2. The categories

will be expanded upon to better understand each reference's contribution. Many of the references provided substantial contribution to areas outside the scope of this research.

Additional Articles Researched	Airlift Metrics	Competitive Advantage	Alliancing	Organizational Culture	Routing	Tabu Search	Value Focused Thinking	Scheduling
Balas and Padberg (1972)					●			
Balinski (1963)					●			
Barnes et al (2004)					●		●	
Becker and Smith (2000)								●
Brigantic and Merrill (2004)	●							
Burke (2004)								●
Burstein (2003)								●
Clay (1989)								●
Dantzig (1958)					●			●
Ferguson (1956)								●
Flood (1955)					●			
Gill (2008)								●
Glover and McMillan (1986)						●		
Gueret et al (2003)					●			●
Hartlage (2012)					●			
Jin et al (2012)					●	●		
Koepke et al (2008)								●
Koskosidis et al (1992)					●			●
Kress and Golany (1994)								●
Morton et al (1995)								●
Nielsen et al (2004)					●			●
Rappaport et al (1991)								●
Rathi et al (1992)								●
Rink et al (1999)					●			
Ruan et al (2011)					●			●
Tryon (2005)							●	
Weir and Johnson (2004)								●
Wilkins et al (2006)								●
Wu et al (2009)					●			

Figure 2: Additional Relevant Sources

Airlift Metrics

Mihram and Nolan (1969) built a stochastic simulation of the strategic airlift system. Their contributions to this dissertation included defining the Strategic Airlift Problem (SAP) and developing the ton miles per day airlift productivity metric. Vincenty (1975) developed an algorithm that calculated an accurate elliptical distance between two points on the Earth. This algorithm was used for all dissertation distance calculations. Yamani (1990) provided a baseline technique for modeling specific aircraft fuel consumption. His model was dependent upon flight at a given altitude. This dissertation expands upon that model and computes fuel consumption at any given flight altitude. Lurdes et al. (1990) and Alexander and Hall (1991) developed pavement classification measures used in the third article of this dissertation.

Lund (1993) supported the assertion that the tons portion of ton-miles was constrained by too few intermediate cargo locations in the Persian Gulf War. Thomchick (1993) also assessed cargo throughput issues associated with the Persian Gulf War. Lewis (1998) analyzed the impacts of Civil Reserve Air Fleet (CRAF) activation due to insufficient military airlift capability. Gendreau and Soriano (1998) evaluated the structural capacity of airfield pavements. This analysis was utilized in the third article of this dissertation. Schmenner (1998, 2001 & 2004) and Babikian, Lukachko and Waitz (2001) demonstrated the relationship between aircraft size and fuel efficiency. The metric selected for their analysis was gallons consumed per available seat mile, which was an inverse of the FEI metric that became a key component of the value model proposed in the second article of this dissertation. Baker, Morton, Rosenthal and Williams (2002) developed a large scale linear programming model for optimizing strategic airlift capability. Their contribution to the metrics portion of this dissertation is in how they measured airfield constraints such as fuel and working Maximum on the Ground (MOG).

Lahiri (2003) proposed metrics for contrasting freight output among differing modes. The airlift metrics selected by Lahiri were air revenue ton miles and air revenue passenger miles. Watson's (2003) contribution to metrics included his definition for an airlift cycle. This definition was incorporated into the value model of the second article of this dissertation. Brigantic and Merrill (2004) in their Algebra of Airlift further validated the use of the selected metrics and provided the equations for the Million Ton Miles per Day (MTM/D) metric. Their algebra makes the assumption that the tons in MTM/D is based on an average payload. This average payload is dependent upon a planning factor that is a weak approximation of reality. This dissertation more accurately models tons in MTM/D by calculating payload capability instead of using a planning factor.

Lee (2004), Hileman et al. (2008), Owen (2008) and Rutheford and Zeinali (2009) developed metrics that included a ratio of energy consumption and output. Lee used Megajoules per Revenue Passenger Kilometers. Hileman et al. used Kilogram Kilometers per Megajoule and called this metric Payload Fuel Energy Efficiency (PFEE). PFEE is the primary contributor to the Fuel Efficiency Index used in the first article. Rutherford and Zeinali used Gallons per Available Seat Kilometers. Sere (2005) and Naylor (2009) suggested distance from origin, parking capacity, fuel capacity, diplomatic relations, proximity to seaports and distance to destination as measures of value when contrasting en route alternatives for airlift routing. Mazraati (2010) highlighted the importance of fuel efficiency metrics by illustrating how fuel costs are increasing as a proportion of airline's total costs. Martin and Voltes-Dorta (2011) used aircraft gross weight as a proxy for the aircraft classification number metric. Oster et al. (2013) analyzed several aircraft safety metrics. Oster's analysis supported a departure obstacle nodal reduction technique in the third article of this dissertation.

Competitive Advantage

Obtaining a sustained competitive advantage is an important goal for many firms. Barney (1991) developed a resource based view of sustained competitive advantage. He suggested that for resources to provide a sustained competitive advantage, they must be valuable, rare, inimitable and non-substitutable. Since fuel is not rare or inimitable, it will not provide for a sustained competitive advantage. Eisenhardt and Martin (2000) adjusted this resource based view to suggest that a firm's dynamic capabilities can achieve a sustained competitive advantage. The first article contributed to this assertion by suggesting that establishing fuel efficiency as a dynamic capability can lead to a sustained competitive advantage for the aviation industry.

Alliancing

Gagnepain and Marin (2007) concluded that airline alliances are able to lower prices because they result in lower costs. Alliancing can be applied to fuel efficiency in that combining requirements can result in higher load factors. Increased load factors result in greater output per energy input. Pooling cargo through alliances to ensure high load factors can be a strategy to enhance fuel efficiency, reduce costs and further establish a firm's competitive advantage.

Organizational Culture

Schein (1984) highlighted the impact of a firm's structure and reward system during the development of organizational culture. Barney (1986) indicated that organizational culture is a potential source for sustained competitive advantage. If a firm's organizational culture is fuel efficiency focused, then that culture has the ability to become valuable, rare, hard to imitate and not easily substituted. Hatch (1993) remarked that changing an organizational culture is hard to achieve.

Routing

Pascal's (1665) triangle models the number of unidirectional routing alternatives given a number of intermediate nodes and a number of en route stops. Pascal's work thus establishes the computational complexity of the strategic airlift problem. His work illustrates the need for nodal reduction techniques to reduce complexity and increase the speed with which valuable alternatives can be analyzed and prioritized. Pascal highlights the value posed by limiting intermediate nodes and en route stops.

According to Flood (1955), the Travelling Salesman Problem (TSP) was first posed by Hassler Whitney in a seminar talk at Princeton in 1934. The TSP attempted to find the shortest routing among a set of nodes ensuring that every node is visited once. The SAP does not require that every node be visited like the TSP, but instead seeks to pick up cargo at a given node and deliver it to a given node with a varying number of intermediate nodes. The TSP is a useful component in the analysis of the SAP, in that given a pickup node and multiple delivery nodes, the TSP can select the optimal delivery route.

Dantzig and Ramser (1958) analyzed routing for the optimal delivery of a fleet of gasoline trucks between a bulk terminal and a large number of delivery stations. The problem ensured all required supplies are delivered while minimizing the total mileage driven. This Vehicle Routing Problem (VRP) with pickup and delivery expands upon the TSP and more accurately models the SAP. Dijkstra's (1959) algorithm attempts to traverse a set of nodes over the minimal cost path and has potential application for edge determination for the VRP as applied to the SAP.

Held and Karp (1962) reduced the computation time of the TSP using dynamic programming. Balinski and Quandt (1963) used an integer programming method on the truck

delivery problem. Mihram and Nolan (1969) developed a stochastic simulation of the strategic airlift system but manually entered 1,820 preferred routes. Balas and Padberg (1972) discussed the VRP as a set covering problem. The sets under consideration are the requirements being delivered. Karmarkar (1984) developed a linear program technique that could be applied to VRPs. Balakrishnan, Chien and Wong (1989) used mixed integer programming with Lagrangian relaxation to obtain a set of optimal alternative routes. They used airline operating profit as the distinguishing criteria between routes.

Bodin (1990) assessed over 20 years of routing and scheduling problems. He discussed several constraints that tend to complicate the problem including multiple vehicle types, vehicle/location dependency, time windows and route length. The SAP has all of these constraints and others. He recommended the use of heuristics to simplify the problem. Yamani (1990) determined the optimal location for aerial refueling of a cargo aircraft given an origin and destination airfield node. Koskosidis, Powell and Solomon (1992) solved the VRP with time windows as a mixed integer program with optimization based heuristics. Instead of hard time windows, they model the problem with soft time windows using early or late penalty costs. Rappoport, Levy, Golden and Toussaint (1992) modeled airlift planning as a vehicle routing problem with time and capacity constraints.

Rink, Rodin, Sundarapandian and Redfern (1999) modeled the routing of airlift aircraft using the double sweep method. They focused on the critical leg distance as the measure to evaluate between routes. Their simplification failed to address aircraft type, capability, crew scheduling, staging and cargo transfer. Cargo transfer eliminates the importance of the critical leg. Baker, Morton, Rosenthal and Williams (2002) optimized airlift using large scale linear programming using the NPS/RAND Mobility Optimizer (NRMO). Their analysis considered

aircraft capability, crew scheduling, airfield parking/servicing constraints, airfield fueling constraints and the transfer of cargo. Routing alternatives for the analysis were entered manually and not created from potential nodes.

Gueret, Jussien, Lhomme, Pavageau and Prins (2003) developed an aircraft loading algorithm and a sortie reduction algorithm based on local search routing manipulation. Bell and McMullen (2004) applied an ant colony optimization technique to the VRP. Barnes, Wiley, Moore and Ryer (2004) solved the Aerial Fleet Refueling Problem using group theoretic tabu search. They considered the problem a multiple vehicle, multiple depot VRP. Crino, Moore, Barnes and Nanry (2004) addressed the theater distribution vehicle routing and scheduling problem using group theoretic tabu search. The primary objective of this multi-modal research was to minimize unmet demand. Nielsen, Armacost, Barnhart and Kolitz (2004) analyzed the channel route structure for airlift using composite variable mixed integer programming. Sere (2005) asserted the importance of diplomatic relations when making routing decisions. Sere's work contributed to a nodal reduction technique in the third article of this dissertation. Longo et al. (2006) provided a transform for turning capacitated arc routing problems into capacitated VRPs. Pisinger and Ropke (2007) introduced the adaptive large neighborhood search heuristic to solve VRPs.

Lambert (2007) addressed the SAP using tabu search. Routing development was based off of a set of previously determined routes and failed to analyze all potentially valuable routes. Chang (2008) addressed route selection for international intermodal networks. Naylor (2009) performed a comparative analysis of the en route system. Wu, Powell and Whisman (2009) merged optimization and simulation for the SAP. Their intuitive technique utilized the stochastic nature and flexibility of simulation while retaining the benefit of the deterministic nature of

optimization. Ai and Kachitvichyanukul (2009) applied particle swarm optimization to solve the VRP. Li, Lam, Wong and Fu (2010) attempted to solve the routing problem using integer programming. Nagata (2010) used a penalty based edge assembly memetic algorithm to solve the VRP with time windows.

Lei, Laporte and Guo (2011) solved the VRP with stochastic demands and time windows. This is an important contribution, since the time phased force deployment list is constantly updated with new requirements. A forecast of those requirements with a given distribution can allow airlift planners to address worst case risk scenarios. Ruan, Zhang, Miao and Shen (2011) combined the vehicle loading and capacitated VRP. Their hybrid approach utilized honey bee mating optimization. Hartlage (2012) utilized the ant colony system for solving the resource constrained shortest path problem. Their work focused on rough cut capacity planning in multi-modal freight networks. Jin, Crainic and Lokketangen (2012) developed parallel multi-neighborhood tabu search for capacitated VRP. Their contribution is to adapt VRP algorithms to the multi-threading capabilities of current processors.

Tabu Search

Tabu search is a method for reducing the computation time for the VRP created by Glover and McMillan (1986). Crino, Moore, Barnes and Nanry (2004) addressed the VRP with tabu search from a theater distribution perspective. Lambert (2007) analyzed the SAP using a tabu search methodology. Jin, Crainic and Lokketangen (2012) adapted tabu search to solve the VRP with parallel processing algorithms. Tabu search is an effective method for solving large scale optimization capacitated VRP problems. The goal of the research of this dissertation is not on the selection of an optimal solution, but on the ranking of a large set of highly valued solutions.

Value Focused Thinking

Keeney (1994) established the concept that alternatives are only valuable in that they can achieve values. Jackson, Jones and Lehmkuhl (1996) applied this Value Focused Thinking (VFT) to airlift in their 2025 future air and space capabilities operational analysis. The measures of airlift value they concentrated on included payload weight, payload volume, range, response and multi-role. Tryon (2005) applied VFT to contingency construction methods. Miravite and Schlegel (2006) also utilized VFT in their Global En Route Basing Infrastructure Location (GERBIL) model that assigned value to different attributes of airfield nodes.

Scheduling

Ferguson and Dantzig (1956) initially devised a linear programming approach to schedule the optimal allocation of a set of aircraft to a set of given routes. Dantzig (1958) scheduled the delivery of bulk fuel from delivery trucks. Samm and Perelli (1982) stressed the importance of circadian rhythm when scheduling. Clay (1989) used temporal constraint propagation to schedule aircraft sorties against a set of requirements. This technique would be useful for meeting airfield operating hours, slot times and required delivery dates.

Bodin's (1990) twenty years of routing and scheduling problems article addressed both the temporal and allocation aspects of scheduling. For the temporal aspect, he noted a lack of adequate research on time windows. For the allocation aspect, he supported the use of heuristics. Rappaport, Levy, Golden and Fashbach (1991) allocated aircraft for scheduling using a hierarchical payload matching procedure. This procedure ordered requirements by latest arrival date, assigned aircraft to requirements by payload maximization, packed planes and then combined requirements from partially filled aircraft.

Rappaport, Levy, Golden and Toussaint (1992) developed an airlift planning heuristic for the scheduling of aircraft. The heuristic utilized the plane preference value for allocation based on payload maximization. Koskosidis, Powell and Solomon (1992) scheduled the VRP with time windows by using soft time window constraints. Rathi, Church and Solanki (1992) built a mathematical model to schedule the airlift of requirements. Their objective function penalized early delivery of cargo, late delivery of cargo and selecting the non-preferred aircraft. Their model considered both port and aircraft capacity constraints.

Kress and Golany (1994) addressed the assignment of aircrews to aircraft in the scheduling process. Morton, Rosenthal and Weng (1995) utilized the General Algebraic Modeling System to schedule aircraft. Time window optimization was handled by cargo specific penalties. Becker and Smith (2000) modeled the allocation of aircraft and aircrews using the Consolidated Air Mobility Planning System (CAMPS) Barrel Allocator tool. An issue facing scheduling using the Barrel Allocator tool is that the allocator cannot optimize his resources for a set of requirements, but instead is given a set of routes with paired aircraft type that he needs to allocate a specific aircraft tail and aircrew against. This often results in inefficient allocation.

Baker, Morton, Rosenthal and Williams (2002) scheduled airlift using large scale linear programming using the NRMO. Time penalties were utilized in the objective function for failure to deliver cargo on time. Burstein, Ferguson and Allen (2003) enhanced scheduling by integrating CAMPS with agent based mixed initiative control. Gueret, Jussien, Lhomme, Pavageau and Prins (2003) developed an algorithm to load the aircraft and then adjust scheduling of aircraft for more efficient loading opportunities. Burke, Love and Macal (2004) analyzed the scheduled deployment of forces and equipment for the Army via truck and rail using the

Transportation System Capability (TRANSCAP) model. Weir and Johnson (2004) used a three phase approach to address the bidline scheduling problem.

Nielsen (2004) scheduled airlift missions for the channel structure using a four step process. First, flight sequences were selected. Using the flight sequence, a generic mission is built adding crew duty day restricted timeline to the flight sequence. Following the creation of the generic mission, cargo is assigned to the single route mission. Finally, multiple routes are examined for cargo aggregation opportunities. Wilkins, Smith, Kramer, Lee and Rauenbusch (2006) described a prototype flight manager assistant to aid in dynamic rescheduling. The article highlights the need for building resiliency into the original schedule. Murphy (2008) stressed the importance of equipment availability at transload locations for scheduling.

Gill (2008) analyzed the capability to schedule a deployment of a Stryker Brigade under several scenarios within 96 hours. The analysis highlighted the limitations associated with hot cargo. Koepke, Armacost, Barnhart and Kolitz (2008) developed an integer program to handle scheduling disruptions caused by working MOG issues. Lei, Laporte and Guo (2011) solve the capacitated VRP with stochastic demands and time windows enabling the scheduling of vehicles. Ruan, Zhang, Miao and Shen (2011) combined the vehicle loading and capacitated VRP. Their technique schedules the optimal cargo allocation across the potential solutions of the capacitated VRP.

III. Original Contribution

The original contribution from the first article of this dissertation includes the aviation industry fuel efficiency model, the association between fuel efficiency and sustained competitive advantage and the analysis of the FEI and its application to the fuel efficiency model. The article in the Journal of Transportation Management provided a comprehensive analysis of fuel efficiency metrics to understand their potential to enable a sustained competitive advantage in airlift. The original contribution from the second article includes the development of an airlift material distance value model and a minimum cutoff distance model. The distance value model utilizes the metrics from the first article. This airlift material distance value model supports the development of the minimum cutoff distance model for airfield nodal reduction

The third article utilizes the cutoff distance model determined from the second article as a nodal reduction and en route stop limiting heuristic. The original contribution of the third article is a set of nodal reduction techniques that can be tailored by the decision maker to eliminate low value airfields before route creation. The third article also assesses the value posed by the various nodal reduction techniques, and examines the effects of combining nodal reduction techniques. The combination of the first three articles sets the stage for rapidly creating a set of high value routes for analysis. Given the set of routes and the metrics from the first article and the value model metrics of the second article, the decision maker can sort routes upon their parameter of focus. In addition, the decision maker will be able to compare and contrast routes on the basis of time, cargo throughput, fuel efficiency and cost.

IV. Journal Articles

Competitive Advantage and Fuel Efficiency in Aviation

Introduction

A firm's efficient utilization of resources can be a source of competitive advantage. For the aviation industry, the resource that makes up the largest component of total cost is fuel. Aviation industry fuel encompassed 20% of total costs in 2007 and United Airlines saw their cost of fuel, as a percentage of total cost, vary between 10% and 25% from 1973 to 2006 (Mazraati, 2010). A dynamic capability to obtain the efficient use of fuel and reduce those costs could lead to a sustained competitive advantage.

Barney (1991) suggests a rationale for a resource based view of sustained competitive advantage. The two main assumptions of this view are that a firm's resources are heterogeneous and that those resources may be immobile across firms. In addition, resources that provide for a sustained competitive advantage must be valuable, rare, inimitable and non-substitutable. Fuel is not rare or inimitable. Fuel as a resource therefore will not provide for a sustained competitive advantage. Yet, a firm's dynamic capabilities properly applied to fuel efficiency can achieve that advantage. Eisenhardt and Martin (2000) expanded upon Barney's resource based view model by adding dynamic capabilities as potential sources of sustained competitive advantage.

Aviation Fuel Efficiency and Dynamic Capabilities

Dynamic capabilities as defined by Eisenhardt and Martin are those "organizational and strategic routines by which firms achieve new resource configurations as markets emerge, collide, split, evolve and die." Some examples given of dynamic capabilities include alliancing, product development and strategic decision making. Eisenhardt and Martin suggest that dynamic capabilities can be a source of competitive advantage by altering a firm's resource base.

The efficient utilization of fuel in the aviation industry is dependent upon alliancing, product development and strategic decision making. A model for implementation of a fuel efficiency strategy can be seen in Figure 3.

Figure 3: Aviation Industry Fuel Efficiency Model

The model's three elements -- strategic decision making, supply chain fuel efficiency and an organizational culture of fuel efficiency directly impact a firm's operational fuel efficiency. Strategic decision making concerning fuel efficiency involves strategic investment and strategic planning. Strategic investment involves the acquisition of aircraft, software, ground equipment and infrastructure improvements. Examples of each of these categories can be seen in Table 1. The critical factor in all of these strategic elements is to consider their fuel efficiency impact on operations. This impact is associated with a purchased item's fuel efficiency and weight. Strategic investments need to consider weight minimization as an important requirement.

Strategic planning involves location management and process decisions. Location management decisions include the basing of aircraft, ground equipment, facilities and maintenance repair capability. The goal of location management is to optimize requirement flow with minimum fuel usage. Process decisions include initial process design for fuel efficiency, process redesign for fuel efficiency and accountability for fuel efficiency. Metrics need to be designed to drive behaviors that increase fuel efficiency in these strategic areas.

Table 1: Aviation Industry Strategic Decision Making for Fuel Efficiency

<table>
<tr><th colspan="6">Strategic Decision Making</th></tr>
<tr><th colspan="4">Strategic Investment</th><th colspan="2">Strategic Planning</th></tr>
<tr><th>Aircraft Acquisition</th><th>Automation and Optimization Software Acquisition</th><th>Ground Equipment Acquisition</th><th>Infrastructure Improvements</th><th>Location Management</th><th>Process</th></tr>
<tr><td>More Fuel Efficient Engines</td><td rowspan="4">Route and Schedule Optimization for Enterprise Requirements at Minimum Cost of Fuel and Assets</td><td rowspan="2">Mission Handling Equipment Fuel Efficiency</td><td rowspan="2">Strengthening a Runway to Increase Load Factors</td><td>Aircraft Basing</td><td rowspan="2">Initial Process Design for Fuel Efficiency</td></tr>
<tr><td>Ligther Materials and Components</td><td>Ground Equipment Locations</td></tr>
<tr><td>Enhanced Aerodynamics</td><td rowspan="2">Mission Support Equipment Fuel Efficiency</td><td rowspan="2">Lengthening a Runway to Increase Load Factors</td><td>Facility Locations</td><td>Process Redesign for Fuel Efficiency</td></tr>
<tr><td>Optimal Fleet Mix for Fuel Efficiency</td><td>Maintenance Repair Capability</td><td>Accountability for Fuel Efficiency</td></tr>
</table>

Supply chain fuel efficiency involves alliancing. Partnering with other firms in the supply chain can result in significant fuel efficiency enhancements. Examples include information technology collaboration that shares aircraft schedules and loads with cargo distribution centers to optimize load factors. Another potential improvement area in alliancing fuel efficiency comes from the increased load factors associated with pooling. Pooling involves sharing requirements to optimize load factors. Gagnepain and Marin (2007) conclude that airline alliances are able to lower prices because they result in lower costs.

Organizational culture is not a dynamic capability, but meets the valuable, rare, inimitable and non-substitutable requirements of a resource based view. Barney (1986) suggests that organizational culture may be a source for sustained competitive advantage. An organizational culture devoted to fuel efficiency is valuable, rare, hard to imitate and not easily substituted. Achieving a fuel efficiency focused organizational culture involves the integration of the importance of fuel efficiency as a core ingredient to the success of the organization. Embedding fuel efficiency into an organizational culture is difficult (Hatch, 1993).

Schein (1984) stressed the importance of the structure of the firm and the firm's reward system during the development of organizational culture. The process to embed fuel efficiency into the culture requires measuring individual contribution to fuel efficiency and then establishing mechanisms that utilize that contribution element as an important consideration for promotion/reward. Leadership involvement is also critical toward embedding fuel efficiency in the organizational culture. Fuel efficiency should be incorporated into leadership communications to employees. Organizationally, a top executive can be assigned to oversee a firm's overall fuel efficiency effort. A committee can also be established among top executives to discuss strategic fuel efficiency opportunities.

Operational fuel efficiency can be greatly enhanced by fuel efficiency strategic decision making, supply chain fuel efficiency and an organizational culture committed to fuel efficiency. To align all of these sources of competitive advantage together requires fuel efficiency metrics. These metrics need to be measured, analyzed and reported to key decision makers. Accountability for metric performance must be established in terms of both individual promotion/reward and fuel efficiency trends needing management attention. The metrics should

be designed to influence positive behaviors and issues where negative behaviors can positively impact a metric should be highlighted and widely acknowledged.

Fuel Efficiency Index

Fuel efficiency metrics in the transportation industry are based upon several aggregate measures of output. In the aviation industry, the Bureau of Transportation Statistics includes air revenue ton miles and air revenue passenger miles (Lahiri et al., 2003). Internationally, revenue ton kilometers and revenue passenger kilometers are used (Owen, 2008). Assuming an increase in these metrics is positive then increasing revenues, distances and load factors would result in a positive trend. The desired objective of fuel efficiency is to move the greatest quantity of cargo and passengers at the least cost of fuel for a given distance, set of assets and unit of time.

Ton miles and passenger miles should measure the Great Circle Distance (GCD) between cargo and passenger onload and offload as established in Federal Regulations (Code of Federal Regulations, 2010). Including GCD in the metric would allow the flight of more miles to save fuel overall. Flying greater distances can save fuel. Examples include flying farther to find more favorable winds or flying farther to obtain an Air Traffic Control routing that allows for a higher, more fuel efficient altitude. Ton miles and passenger miles still fail to take into account fuel, so those metrics should be divided by fuel used. The literature includes many examples where fuel is incorporated with passenger distance and cargo weight distance (Lee et al., 2004; Hileman et al., 2008; Owen, 2008; Rutherford and Zeinali, 2009). Ton miles per lbs of fuel consumed and passenger miles per lbs of fuel consumed consider fuel and mass transported over a given distance.

Hileman et al. (2008) labeled these metrics Payload Fuel Energy Efficiency (PFEE), but uses fuel energy consumed instead of lbs of fuel consumed. This metric excels as an aggregate

measure, but fails to take into account how an increasing quantity of sorties can tend to increase the measure of efficiency. For example, if two sorties are performed exactly the same, then the aggregate PFEE of both sorties is twice the size for the PFEE of one sortie. The reason for this is that both variables in Hileman et al.'s metric numerator are doubled while only one term in the denominator is doubled. This effect of increasing efficiency by increasing sorties is eliminated by obtaining the sortie average. Including the number of sorties *n* in the denominator of PFEE operationalizes the Fuel Efficiency Index (FEI) metric as seen in Equations 1 and 2.

$$Cargo\ FEI = \frac{\sum_{i=1}^{n} \frac{Tons_i * Miles(GCD)_i}{KLbs\ of\ Fuel\ Burned_i}}{n}, where\ n = \#\ of\ sorties \quad (1)$$

$$Passenger\ FEI = \frac{\sum_{i=1}^{n} \frac{Passengers_i * Miles(GCD)_i}{KLbs\ of\ Fuel\ Burned_i}}{n}, where\ n = \#\ of\ sorties \quad (2)$$

The Data

Babikian et al. (2001) demonstrated that efficiency differences between regional and large aircraft can be affected by sortie length. As the proportion of large and small aircraft changes over time, the overall FEI can be biased. To remove this bias, the FEI in Equations 1 and 2 can be calculated on an aircraft type basis to remove the bias of different aircraft type ratios impacting the overall efficiency metric. To obtain a better understanding of the fuel efficiency index, 5,144 Air Mobility Command military airlift sorties from November 2010 were analyzed with respect to the proposed index. Only channel, contingency or special assignment airlift mission sorties were selected. A summary of the index numbers broken down by aircraft Mission Design Series (MDS) can be seen in Table 2.

Table 2: Air Mobility Command FEI by MDS November 2010

	Sorties	Great Circle Distance (Nautical Miles)	Cargo (Tons)	Fuel Consumed (1000 lbs)	Fuel Efficiency Index: $\frac{\sum\frac{GCD*Cargo}{FC}}{Sorties}$
C-17A	3,110	4,471,385	54,406	220,724	327
C-5A	74	133,192	1,782	8,141	387
C-5B	251	542,520	7,494	31,936	478
C-5M	4	10,375	116	549	571
C-130E	317	64,456	861	1,661	101
C-130H	675	280,850	2,563	6,492	148
C-130J	188	145,918	831	2,587	257
KC-10A	107	186,420	289	14,955	68
KC-135R	358	494,280	459	26,663	53
KC-135T	60	74,927	49	5,265	33
Total	5,144	6,404,322	68,850	318,971	Average: 267

Note how the larger aircraft tend to have on average better FEI scores with the C-5M scoring highest. This trend for larger aircraft matches Babikian et al.'s results. Tanker aircraft (KC-10 and KC-135) tend to have very low FEI scores due to the limited cargo they carry and also due to the fact that airlift is ancillary to their primary mission of air refueling. Their overall efficiency numbers are at the lower end of their range due to the prevalence of sorties with no cargo. Of all the sorties observed, 22% had no cargo. Sorties at the top of the efficiency range had FEI measuring in the thousands. Table 3 includes the descriptive statistics for all of the FEIs.

Table 3: Descriptive Statistics for Air Mobility Command FEI November 2010

Mean FEI	267
Standard Deviation	332
Minimum	0
Maximum	5,189
Count	5,144

From the descriptive statistics, note that the standard deviation is larger than the mean. This suggests a large dispersal of the data. There are a few outliers at the top of the range that are associated with bad data. A couple of cases included diverts back to the origin, but failed to change the city pair. This resulted in extremely low fuel usage for a long distance resulting in a false FEI. In the cases of diverts, it is important to record the destination as the same as the origin. Finally, the mean is much larger than the median suggesting influence by a few outliers at the top of the range.

Great Circle Distance

After examining the descriptive statistics of FEI, the data was analyzed to assess the impact of great circle distance. If greater distances lead to better FEIs, than shifting the fleet to more long distance missions might improve the FEI measure. Increased distance tends to decrease payload capacity. This can be seen in Breguet Range Equation 3 (Lee et al., 2004). V is the flight speed, L/D is the lift to drag ratio, g is the gravitational acceleration constant, SFC is specific fuel consumption and W is weight. The equation shows a tradeoff between fuel weight and payload weight.

$$R = \frac{V\left(\frac{L}{D}\right)}{g*SFC} * \ln\left(1 + \frac{W_{fuel}}{W_{payload} * W_{structure} * W_{reserve}}\right) \quad (3)$$

If AMC aircraft were operating at maximum payload, then as distance increases, payload decreases counteracting the increase in FEI. When not operating at maximum payload, similar payloads will result in a higher FEI for aircraft that move the cargo farther. To isolate the bias of differing MDS aircraft, the comparison of distance to FEI was made for the C-17 and the C-5.

For the C-5, the A, B and M models were included together. The results were plotted in Figures 4 and 5.

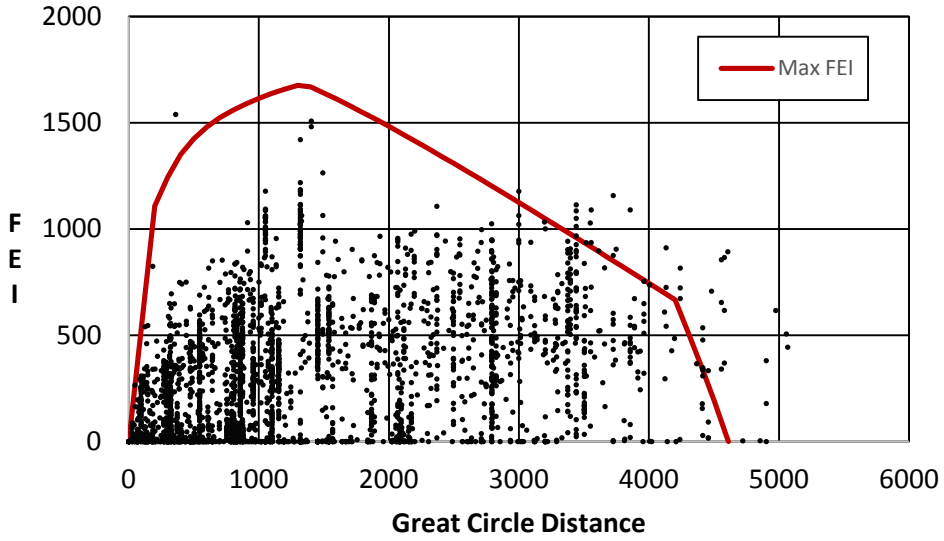


Figure 4: C-17 Great Circle Distance and FEI

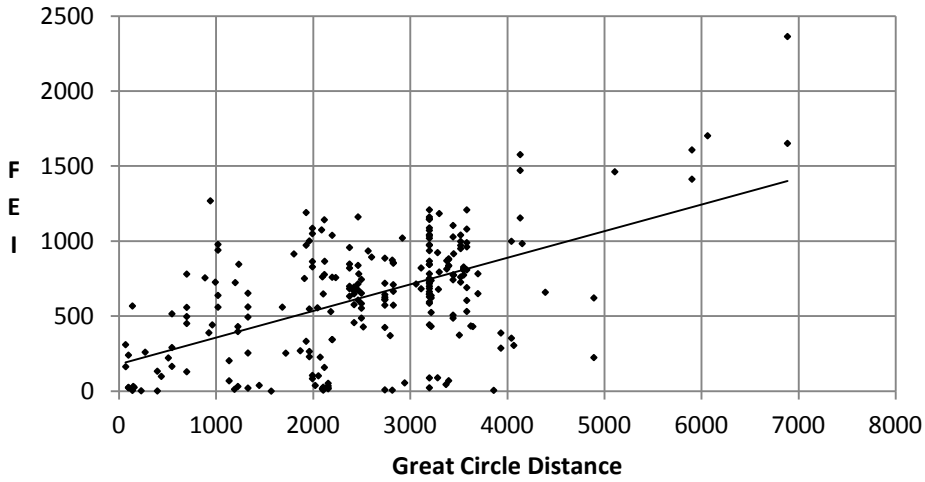


Figure 5: C-5 Great Circle Distance and FEI

Figure 4 shows the maximum FEI for the C-17 in red in addition to the specific sorties. This maximum FEI was calculated using zero wind and assumes that alternate and holding fuel

is the same as reserve and contingency fuel. Varying winds and fuels explain why certain sorties in Figure 4 are able to exceed the maximum FEI. Note the peak in max FEI in Figure 4 for the 1,000 to 2,000 NM range. Air refueling sorties were removed from Figure 4 to better highlight the relationship between the sorties and max FEI.

Both Figure 4 and 5 show an increase in FEI for longer distance city pairs. The overall correlation between GCD and FEI is 44%. The only method that a manager could use to increase GCD is to overfly an intermediate location or discover longer distance city pairs to replace city pairs currently being used. If these sorties were operating at maximum payload before the transition, than a payload penalty would exist for going to longer distances. Yet, if the sorties were flying with a suboptimal payload, then they could fly a longer range with the same payload and increase FEI.

Load Factors

To enhance the effectiveness of the FEI, it should be reported along with load factors. The benefit of the load factor is that it is a ratio of the actual load to the optimal load. This information provides important insight into how cargo loading efficiency influences FEI. Load factors can have two limiting factors. These factors include weight limitations and volume limitations. The volume limitation or cube is a matter of dimension. It is based on the surface area of the cargo floor and the height of the cargo door. It is often measured as a ratio of pallet positions used over pallet positions available. If a cargo compartment is cubed out (pallet positions used equals pallet positions available) and cargo of greater density is not available (assuming below payload maximum) then the horizontal optimal configuration was achieved. In order to achieve optimality for the vertical, a metric should be added for the load factor of the pallet. It should be noted that calculating pallet load factors could be complex if accuracy is a

primary concern. To simplify pallet load factors, a ratio of the height of the pallet to the maximum allowable height might be preferable.

The weight limitation is more complex. Pallets and aircraft cargo floors have a weight limitation. The limits of these must be observed. The aircraft also has a maximum gross takeoff weight which is dependent upon several variables. The first constraint is an airframe limit. This airframe limit can be reduced based upon several variables. These variables include pavement strength, runway length, altitude, temperature, obstacles and runway winds. With the maximum gross weight for takeoff determined, cargo available equals maximum gross takeoff weight minus operating weight minus fuel on board. The fuel on board is a calculation based on many factors.

The primary factor is the distance to the next fueling point. Other considerations include icing, thunderstorms, weather at origin and destination, distance to alternate, airframe specific fuel degrade, cargo weight, routing, altitude and winds. Due to the complexity of all of these factors, determination of the exact maximum payload is extremely difficult and often requires iterative algorithms. Computer flight planning software can calculate the value of payload maximum (PMAX) and those values should be calculated and recorded for every sortie flown. For passengers, the load factor is based on percentage of seats filled. See Equations 4 through 8 for load factors. The behaviors desired from these metrics include maximizing the pallet loads and completely filling the aircraft.

$$Pallet\ Load\ Factor\ (Cube) = \frac{Actual\ Pallet\ Volume}{Maximum\ Pallet\ Volume}\ or\ \frac{Actual\ Pallet\ Height}{Maximum\ Pallet\ Height} \tag{4}$$

$$Pallet\ Load\ Factor\ (Weight) = \frac{Actual\ Pallet\ Weight}{Maximum\ Pallet\ Weight} \tag{5}$$

$$Load\ Factor\ (Cube) = \frac{Pallet\ Position\ Equivalents\ Used}{Pallet\ Positions\ Available\ (MDS\ Specific)} \tag{6}$$

$$Load\ Factor\ (Weight) = \frac{Actual\ Cargo\ Weight}{Computer\ Flight\ Plan\ computed\ Payload\ Maximum} \quad (7)$$

$$Passenger\ Load\ Factor = \frac{Actual\ Passengers}{Available\ Seats} \quad (8)$$

Load factors for passengers in the aviation industry grew from 60 to 80% from 1990 to 2008 and load factors for commercial cargo remained flat around 60% over the same time period (Hileman et al., 2008). To contrast against industry data, load factors for the Air Mobility Command data set were gathered. Payload maximum was determined using Equation 9. Actual ramp fuel was used to aid in simplification, but operationally the load factors need to be determined before the ramp fuel is loaded. Payload maximum is not routinely used by Air Mobility Command's command and control staff, but its value is critical to accurate load factor determination during planning.

$$Payload_{Max} = Min(Payload_{Acft\ Max}, (W_{Max\ Gross\ Takeoff} - W_{Ramp\ Fuel} - W_{Operating})) \quad (\textbf{Error! Bookmark not defined.})$$

Payload maximum is dependent on Maximum Gross Takeoff Weight. For the analysis, the Maximum Gross Takeoff Weight used was the maximum for the aircraft. Other variables that could further reduce Maximum Gross Takeoff Weight include airfield pavement strength limitations and departure obstacles. Their inclusion would serve to improve load factors. The cargo load factors for Air Mobility Command can be seen in Table 4. The Air Mobility Command cargo load factor is lower than industry by a factor of 3. This illustrates the need for the operationalization of the load factor metric into Air Mobility Command planning, command and control. Each sortie's load factor needs to be highlighted when the value falls below a firm's

specific threshold. Load factor feedback control systems can have a positive impact on the fuel efficient operation of the enterprise.

Table 4: Air Mobility Command Load Factors November 2010

	Maximum Gross Takeoff Weight	Empty Weight	Load Factors
C-17A	585	282.5	23%
C-5A	769	380	23%
C-5B	769	380	31%
C-5M	769	380	28%
C-130E	155	90	15%
C-130H	155	90	21%
C-130J	155	90	27%
KC-10A	590	241	3%
KC-135R	322.5	119.23	3%
KC-135T	322.5	119.23	2%
Total			22%

Strategic airlift airframes were selected from the data for more detailed analysis. To better understand the impact of load factors on FEI, load factors were plotted against FEI for both the C-17 and the C-5 as seen in Figures 6 and 7. In both cases, a positive correlation is seen between increasing load factors and the FEI. Overall, there exists a 74% correlation between load factor and FEI. This is almost twice as large as the 44% correlation with GCD. There are several data points outside 100% load factors. These are suspected to be due to waivers that allow for loading more cargo than Maximum Gross Takeoff Weight. One other item of note is the increasing variance of FEI as load factors increase. This was also apparent in the analysis of GCD.

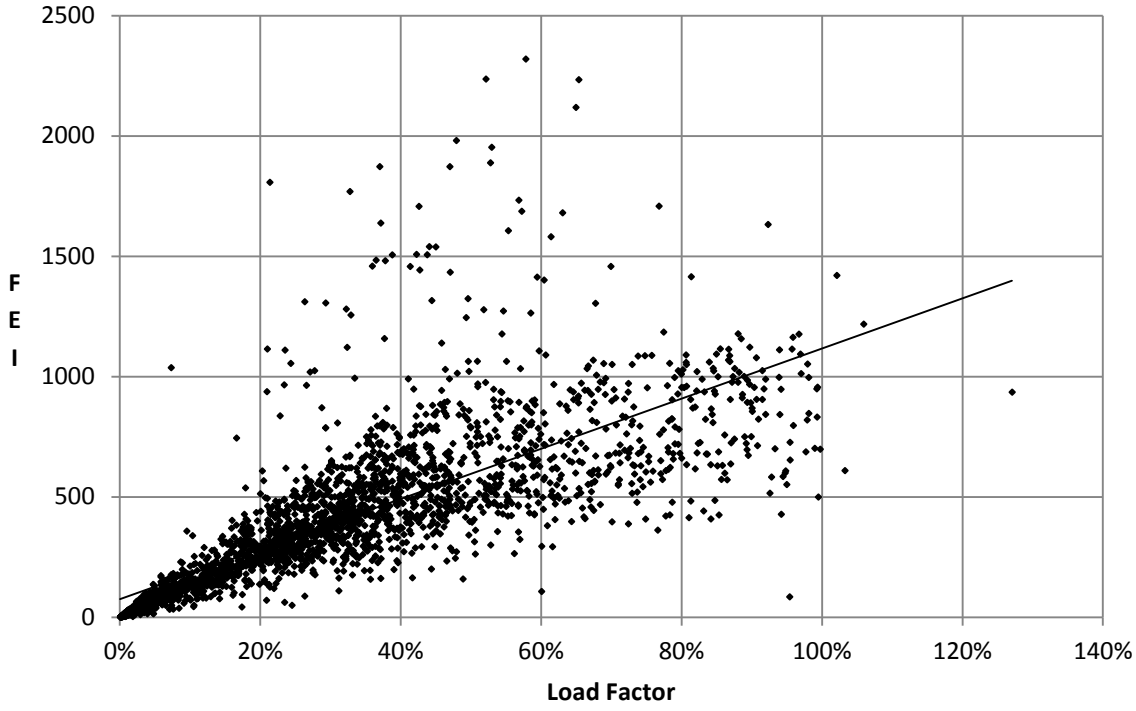


Figure 6: C-17 Load Factor and FEI

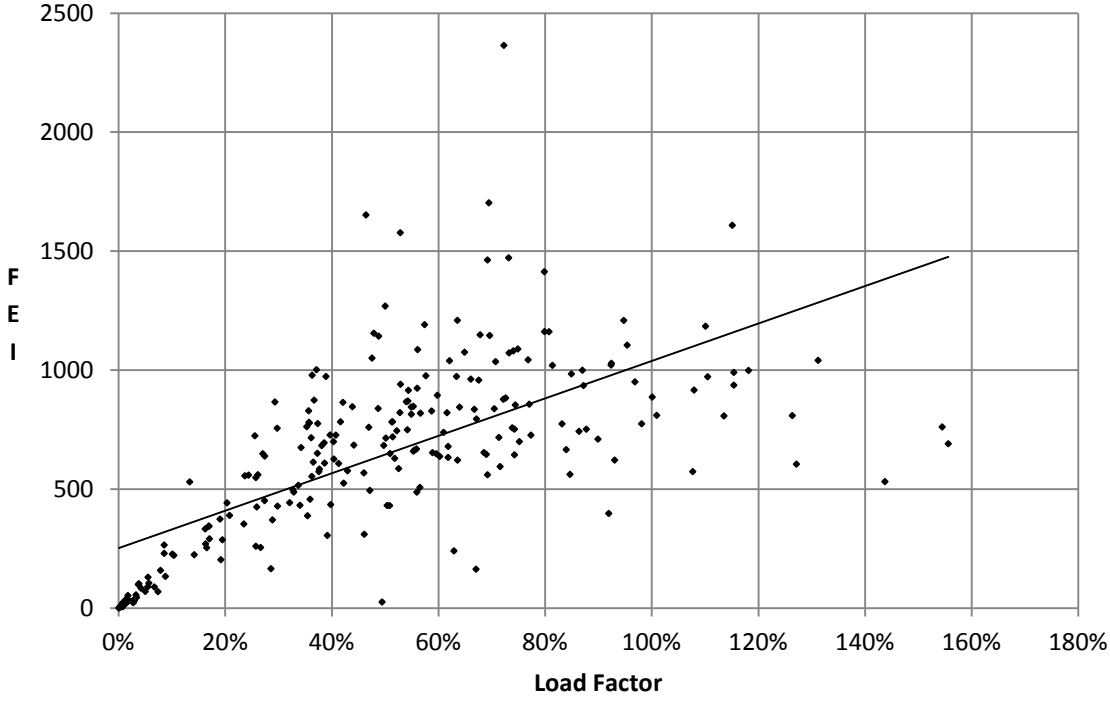


Figure 7: C-5 Load Factor and FEI

Inactive Sorties

Aircraft need to often position to pick up cargo and deposition after delivering cargo. This reduces load factors by driving up the number of no cargo sorties. It also reduces FEI due to the zeroing of the numerator. Inactive sorties drive the desire to either stage aircraft out of heavy cargo and passenger requirement locations or to select aircraft that are nearest to the cargo and passenger requirement onload or offload locations. A metric that is proposed to handle the efficiency of aircraft selection to meet this requirement is inactive miles per inactive sortie as seen in Equation 10. An inactive mile is defined as a mile flown to position an aircraft at a cargo on load location or to deposition an aircraft from a cargo offload location. An inactive sortie is a sortie composed of inactive miles. The behavior desired is to drive aircraft staging to where the cargo is located or to select an aircraft for a mission that is closest to the cargo on load and offload.

$$Inactive\ Miles\ per\ Sortie = \frac{\sum_{i=1}^{n} Inactive\ Miles_i}{n},$$
$$where\ n = \#\ of\ inactive\ sorties \tag{10}$$

The results of the inactive miles per sortie analysis on an MDS basis for Air Mobility Command can be seen in Table 5. The tankers have to travel the longest to get their requirements. Inactive miles appear to decrease with aircraft size after that. Although this metric is broken down on a per MDS basis, it could be analyzed on a departure airfield basis to discover which units have the farthest to travel for positioning and depositioning. From these results, insights into potential staging opportunities could be an area for further research.

Table 5: Air Mobility Command Inactive Miles Per Sortie November 2010

	Inactive Sorties	Inactive Miles	Inactive Miles Per Sortie
C-17A	960	1,186,113	1,236
C-5A	33	27,453	832
C-5B	98	129,808	1,325
C-5M	2	5,188	2,594
C-130E	40	18,876	472
C-130H	49	47,441	968
C-130J	31	29,748	960
KC-10A	37	88,638	2,396
KC-135R	77	163,989	2,130
KC-135T	7	7,493	1,070
Average			1,398

Fuel

After examination of the effects of Great Circle Distance and Load Factors on FEI, the final variable that is part of FEI is fuel consumed. An examination of fuel consumed against FEI was plotted in Figures 8 and 9. To aid in visibility for the C-17 plot, three outliers were removed. The expected behavior is that as fuel consumed increases, FEI should decrease. The opposite occurs in actuality. There are two suspected reasons for this. First, there is a 78% correlation between GCD and fuel consumed and the FEI increase associated with increasing GCD outweighs the additional fuel burned. Second, sorties with higher load factors burn more fuel. A potential solution to provide greater sensitivity to fuel consumed would be to square the fuel consumed in the denominator of the FEI equation.

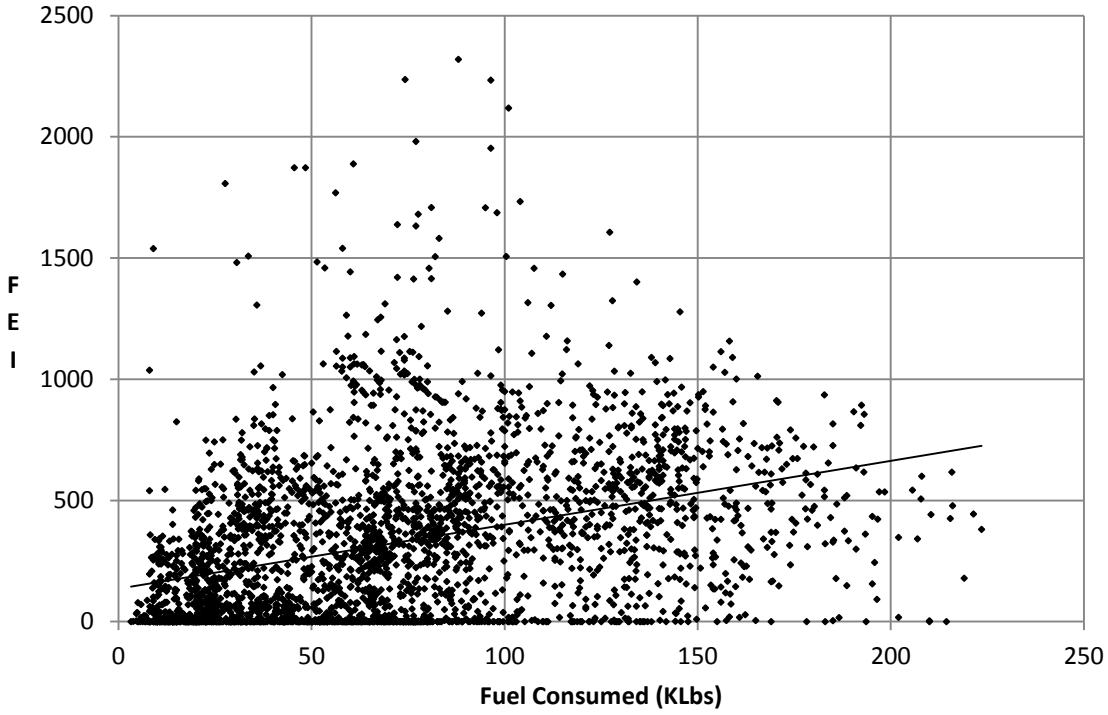


Figure 8: C-17 Fuel Consumed and FEI

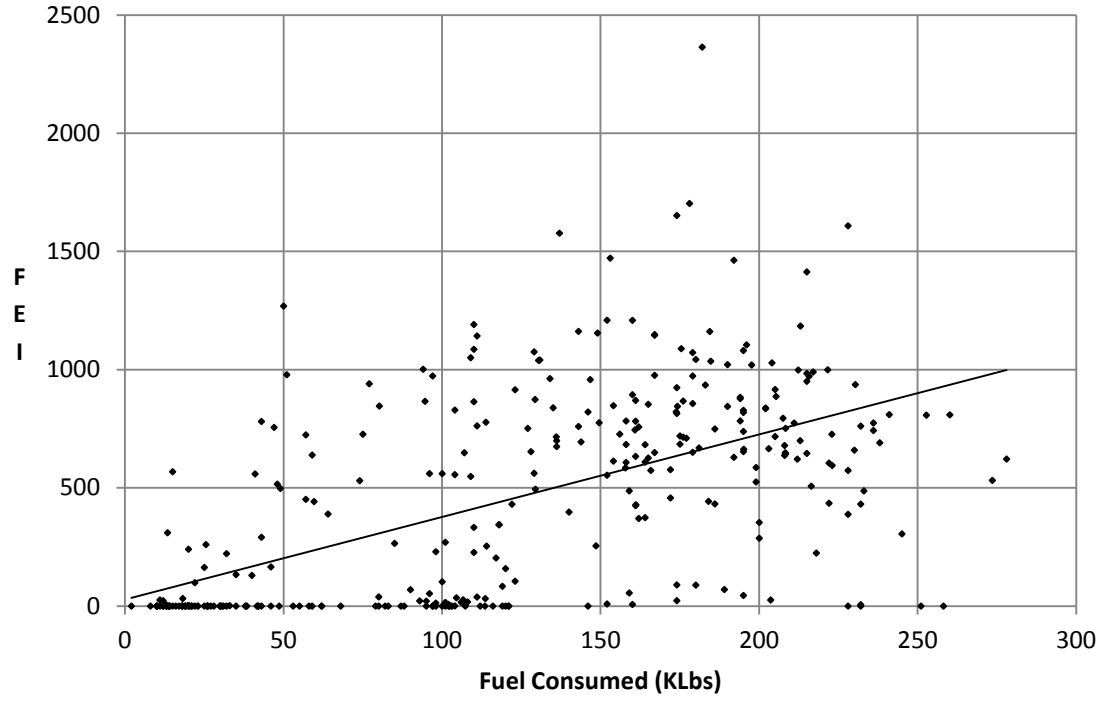


Figure 9: C-5 Fuel Consumed and FEI

When extra fuel is carried on board an aircraft, the added weight of that fuel burns additional fuel unnecessarily. Due to this cost of carrying additional fuel, it is often desired to ensure that no more fuel is added to a mission than planned. This illustrates the need for a metric that represents fueling accuracy as seen in Equation 11. In addition to reducing the cost to carry fuel, it is often desired to have the aircraft fly the most fuel efficient flight profile. This is complicated by load factors and distances involved. To remove these and other sortie specific factors, a contrast could be made between a planned fuel burn and the actual fuel burn. To drive this behavior, Equation 12 measures a planned over actual fuel burn ratio. The goal of the metric is to maximize the ratio by minimizing actual fuel burn.

$$Fueling\ Accuracy = Max\left(0, Min\left(1{,}1 - \frac{actual\ fuel\ load - planned\ fuel\ load}{planned\ fuel\ load}\right)\right) \quad (11)$$

$$Fuel\ Burn\ Ratio = \frac{planned\ fuel\ burn}{actual\ fuel\ burn} \quad (12)$$

Differences between planned and actual fuel burn are subject to multiple variables. Many of these variables are outside of the pilot's control while some can be manipulated. Variables outside of the pilot's control include winds different than planned, achievable altitude below planned, icing/thunderstorms/turbulence altering routings and/or altitude and decreased engine performance. Variables within the pilot's control include throttle setting, not flying planned routings and altitudes (not influenced by external constraints) and climb/descent profiles. Since the ratio of planned fuel burn to actual fuel burn does not distinguish between aspects of fuel burn that are within the pilot's locus of control, the metric could be unjustly punitive. Despite this drawback, the metric does distinguish discrepancies from planned fuel burn and drives

behavior to lower fuel burn. Air Mobility Command data for average fueling accuracy and average fuel burn by aircraft can be seen in Table 6.

Table 6: Fueling Accuracy and Fuel Burn Ratio

	Average Fueling Accuracy	Average Fuel Burn Ratio
C-17A	97%	1.03
C-5A	95%	0.98
C-5B	98%	0.98
C-5M	100%	1.02
C-130E	100%	1.00
C-130H	99%	1.01
C-130J	93%	1.11
KC-10A	96%	0.98
KC-135R	92%	1.00
KC-135T	97%	1.00

From the table, note the high fueling accuracies. These high accuracies are due to the way the planned ramp fuel is calculated. The Air Mobility Command Fuel Data Tracker will set the planned ramp fuel equal to actual ramp fuel if the ramp fuel deviation reason was outside of the pilot's control. This aids in unjust attribution, but skews the data toward the high end of accuracy. The fuel burn ratio provides little information from an aircraft perspective. It might suggest something about the quality of the fuel planning or it could be a sign of something cultural in that aircraft's community. The fuel burn ratio could be more effectively used by comparing organizational units. It could also be used to compare pilots.

Managerial Implications for City Pair Analysis

FEI increased with GCD, load factor and fuel consumed. To get a better understanding of the sensitivity of FEI to load factor and fuel consumed, a specific city pair was selected. This

enabled distance to become constant leaving cargo and fuel as the remaining variables. Dover to Ramstein was a common city pair in the data set with 20 observations. Note that managing FEI by city pair might be time consuming and effort should be concentrated on frequent city pairs. C-17s were selected for the analysis to further constrain the variables by limiting aircraft type. The results can be seen in Figures 10 and 11.

Figure 10 shows how the amount of fuel consumed varies for a fixed distance and load factor, while Figure 11 shows how the amount of cargo varies for a fixed distance and fuel consumed. The Figure 10 relationship is useful for managers in that it identifies sorties that deviate from previous observations based on fuel efficiency. The ability to identify sorties that exceed a predetermined interval on the regression of that city pair could highlight outliers in both fuel efficiency and fuel inefficiency. In depth analysis of those outliers in terms of root cause could expose opportunities for greater fuel efficiency. Specific aircraft tails or aircrews might repeatedly occur outside the interval representing the need for possible remedial action.

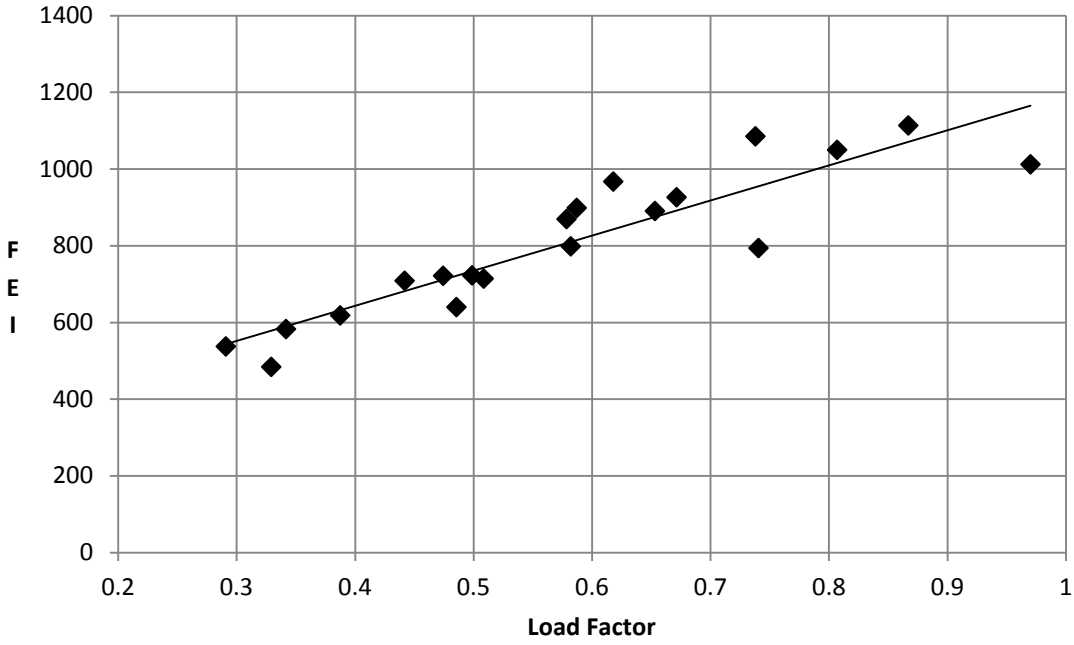


Figure 10: KDOV-ETAR C-17 Load Factors and FEI

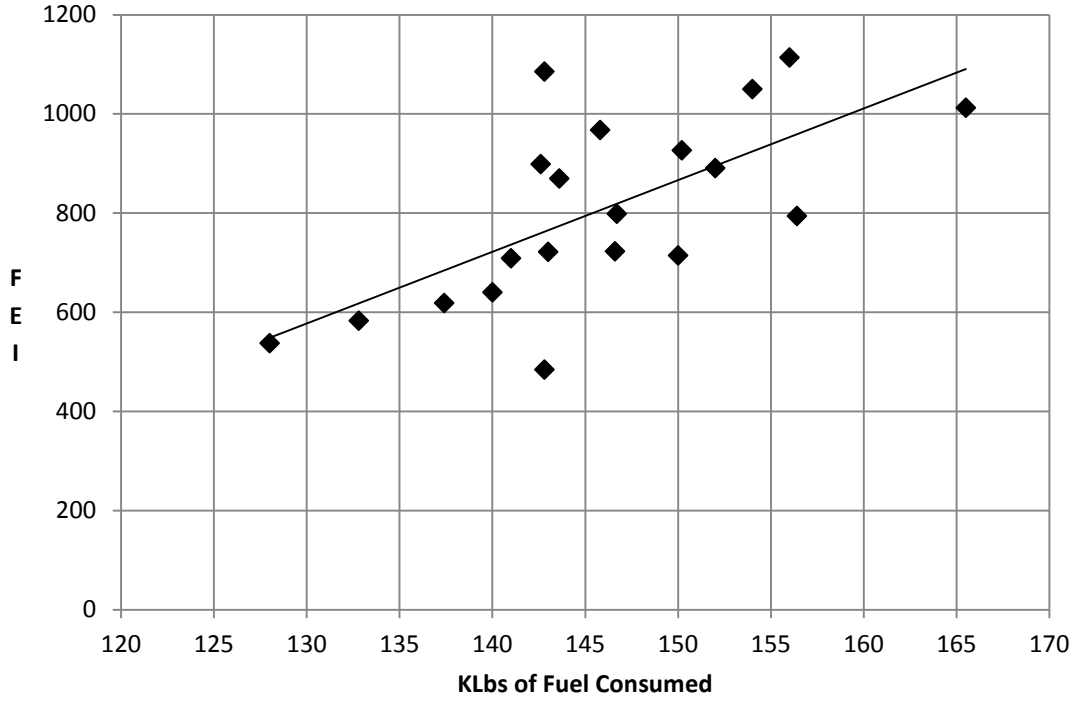


Figure 11: KDOV-ETAR C-17 Fuel Consumed and FEI

From Figure 10, note the tight scatter of points about the simple linear regression. The R^2 for this regression is .82. This indicates that load factor when constricted by city pair explains most of the variability in FEI. Figure 10 also aids in understanding that to target an FEI near 1,000 requires an 80% load factor. From Figure 11, note that the points have much greater variance about the line. The R^2 for this regression is .45. This indicates that fuel consumed when constricted by city pair explains only 45% of the variability in FEI. Taking a vertical slice of Figure 10 shows load factor replicates with the variance between the data points being explained by fuel consumed. Using a band about the regression line for a city pair in Figure 10 could highlight missions that consume too much or too little fuel contrasted against the aggregate. Further analysis into those missions could potentially highlight fuel savings opportunities.

Incorporating Metrics into the Aviation Industry Fuel Efficiency Model

Application of FEI operationally can drive desired behaviors to increase load factors, reduce inactive miles and reduce fuel usage. Reducing fuel consumption might best be addressed through the banding method of the regression line in the Dover to Ramstein example. FEI has value beyond application operationally. To obtain the optimal value from FEI, the metric should be applied to all of the components of the Aviation Industry Fuel Efficiency model. The first component of the model requiring the application of FEI is strategic decision making. FEI should be implemented in both the strategic investment and strategic planning components of strategic decision making,

From a strategic investment perspective, the FEI metric can drive aircraft acquisition requirements and allow for innovative paradigm shifts. The FEI minimum for several set distances can be specified as the requirement. Since FEI does not include time as a variable, that

should be constrained to a set maximum when building the requirement to avoid solutions that are too slow. FEI also fails to address reliability. The C-5 has superior FEI on average, but suffers from reliability issues. This needs to be addressed when making strategic investments such as aircraft acquisition. Larger aircraft might be superior in terms of FEI, but might suffer mechanically due to their size and complexity. Infrastructure improvements enhancing load factor potential such as pavement strengthening can be assessed based upon FEI impact. Strategic airfield improvements could result in increased cargo flow and more efficient operations. Ranking airfield improvement projects by FEI impact can be an important factor when considering prioritization.

Beyond strategic investment, FEI could be extremely useful in strategic planning. FEI and inactive miles would be useful for the determination of aircraft basing and staging locations. Those metrics would also be useful from a theory of constraints perspective by highlighting the least efficient aircraft and mission pairings. Automatically calculating the FEI planning metric once an aircraft was assigned to the mission and highlighting poor FEIs and inactive miles could provide planning and aircraft allocation immediate feedback for correction. Individual planners and aircraft allocators can be held accountable using FEI and inactive miles as performance metrics. Beyond individuals, organizational goals can be established regarding both the FEI and inactive miles.

Implementation of the FEI should extend beyond the firm when the FEI is dependent upon other firms in the supply chain. Suppliers performing functions such as warehousing and distribution that are tied to air mobility should be provided information on their FEI impact. In addition, strategic partnering should be encouraged to enhance load factors. Alliances should be examined that offer the greatest potential to increase the FEI. Shared investments on information

technology, automated identification and tracking and cargo distribution equipment might offer FEI improvements that justify the acquisition. Suppliers need to be properly rewarded for their investments to enhance FEI.

Strategic decision making and supply chain fuel efficiency can be greatly improved through the use of the FEI. Yet, there are areas of improvement in FEI that can only be achieved by those operational workers executing the process. To reap those benefits, FEI needs to be embedded into organizational culture. Attempting to embed a metric into organizational culture and simultaneously using the metric as a tool for accountability is difficult. The problem is that individuals tend to rebel against punitive metrics. For acceptance, it is preferred to use the metric in a positive role until it becomes accepted as part of the organization. It is important to include the metric when measuring operations at every level. Obtaining leadership support for the metric is essential. FEI needs to be presented at senior level meetings and included in organizational goals. Finally, FEI should be part of the reward structure for promotion for factors within the individual's control. This could include individual awards for sustained high FEI performance to highlighting the metric during promotion discussions.

Findings and Conclusion

The Aviation Industry Fuel Efficiency model presents a framework for transforming fuel efficiency into a sustained competitive advantage. This is achieved through the use of the dynamic capabilities of strategic decision making and alliancing. In addition to those dynamic capabilities, the model recommends ingraining fuel efficiency into the organizational culture. To assist the manager in implementing the model, the FEI was introduced. The FEI drives desired behaviors to increase load factors, decrease inactive miles and reduce fuel consumed. Other metrics were suggested to further assist the manager in improving fuel efficiency behaviors to

include load factors, inactive miles per sortie, fueling accuracy and fuel burn ratio. It is important to measure load factors from both a weight and cube perspective, to obtain a better understanding of the efficiency of operations.

Measuring FEI operationally can drive behaviors toward increased fuel efficiency, but application of the FEI to the model is where a firm can leverage much greater fuel efficiency benefits. Extending the FEI to strategic decision making, supply chain partners and the organizational culture will allow the firm's fuel efficiency focused resources to not be easily imitated. There are certain risks associated with greater fuel efficiency integration within the supply chain and strategic fuel efficiency investments. These risks need to be thoroughly analyzed. There are also risks to not integrating or not investing in an environment of rising fuel prices. Following a fuel efficiency strategy will make the firm and the firm's supply chain less susceptible to rising fuel prices. A fuel efficiency strategy will also increase a firm's ability to compete on price.

The FEI ties together all of the components of the model. It enables individual, organizational, corporate, supply chain and industry goals to align. This common sense of purpose can only be achieved if the metric is valued equally. FEI could support aircraft manufacturers, distribution centers, command information systems, planning systems and allocation. Much as a low cost retailer is less susceptible to economic downturns, a fuel efficient firm in the aviation industry is less susceptible to fuel price increases. A fuel efficiency strategy is a risk reduction strategy with opportunities for expert practitioners to obtain a sustained competitive advantage.

Distance Value Model for Nodal Reduction of the Strategic Airlift Problem

Introduction

Global airlift operations require an analysis of an extremely large set of airfields. The problem of selecting the optimal airfields from this set to service a customer airlift requirement is an essential component of the Strategic Airlift Problem (SAP). Mihram and Nolan (1969) define the SAP as "a set of time-phased movement requirements for troops, vehicles and dry cargo, each constituting a demand on airlift resources." The problem of optimally selecting the set of time-phased movements of the SAP is a class of routing and scheduling problems commonly referred to in the literature as the Vehicle Routing Problem (VRP). The most basic VRP contains a single depot node and several other nodes denoting customer delivery locations. These locations are connected by edges (arcs) that represent the minimum cost path between nodes. The goal of the VRP is to find a set of routes from the depot to the customers that minimizes the total cost to meet the customers' demand. The edge cost can be calculated in many ways but most frequently it is simply the minimum distance path between nodes, see for example Bell and McMullen (2004), Longo et al. (2006), Pisinger and Ropke (2007), Chang (2008), Ai and Kachitvichyanukul (2009) and Nagata et al. (2010).

Using the minimum distance for an aircraft flight between two airfields can be problematic. If the edge distance is large enough, then payload is sacrificed for the additional fuel needed. To increase the amount of the payload, an intermediate stop, called an en route stop, can be made. Lambert (2007) defined the SAP such that given a set of materiel and personnel requirements and their associated on-load and off-load locations, specific aircraft will be assigned to load those requirements and transport them to their off-load locations. Aircraft can choose to fly directly from the requirement on-load to off-load locations, stop at an en route

location for fueling, or use an en route cargo location for trans-load operations. That routing decision on whether to go direct, stop for gas, or trans-load is complicated by the interaction between fuel, payload, and distance.

Thus a sub-problem emerges with respect to airlift. Given a set of airfields between requirement pickup and delivery, what is the optimal path to reach delivery? In this sub-problem, nodes are the airfields that serve as en route stops between the pickup and delivery locations. Edges are defined as the sorties (flights, or trips) between airfields and paths are the set of edges from pickup to delivery. While most research on the VRP focuses on minimizing the total cost of transportation given the edge costs, this research focuses on how to build the set of edges for the VRP. The further use of the terms nodes, edges and paths will refer to this sub-problem. Bodin (1990) reflected on 20 years of routing and scheduling problems and detailed several constraints that tend to complicate the problem including multiple vehicle types, vehicle/location dependency, time windows and route length. All of these problems also exist in our sub-problem of edge determination.

Therefore, the goal of this research is to decrease the time to select the optimal aircraft path for a given segment of the VRP from pick up to delivery. The selection of this optimal path for a given requirement is a necessary condition to establish the capacity constraint of the capacitated vehicle routing problem. The method chosen to obtain optimality faster is nodal reduction. We seek to eliminate nodes that are of such low value that they would not be part of the optimal solution. Can airfield nodes be removed through value modeling without eliminating optimal or highly desirable solutions and what improvement in speed can be obtained through this nodal reduction?

To answer these questions, we begin by developing basic assumptions. The first assumption is that any en route airfield selected is closer to the origin and destination than the origin-destination distance. This assumption is captured in Equations 13 and 14 and can be seen in eye shape of Figure 12. Miravite and Schlegel (2006) call this eye shape the "Lens." This assumption provides for unidirectional aircraft flow and reduces computation time from $O(e^{n!})$ to $O(2^n)$. We will later show that there is one potential flaw to this basic assumption due to airfield pavement strength that needs to be addressed and offer a potential solution.

$$\delta_{OE} < \delta_{OD} \quad (13)$$

$$\delta_{ED} < \delta_{OD} \quad (14)$$

Where:

δ_{OE} = Origin-En route Distance
δ_{OD} = Origin-Destination Distance
δ_{ED} = En route-Destination Distance

Figure 12: Dover airfield to Ramstein airfield "Lens" (GCmapper)

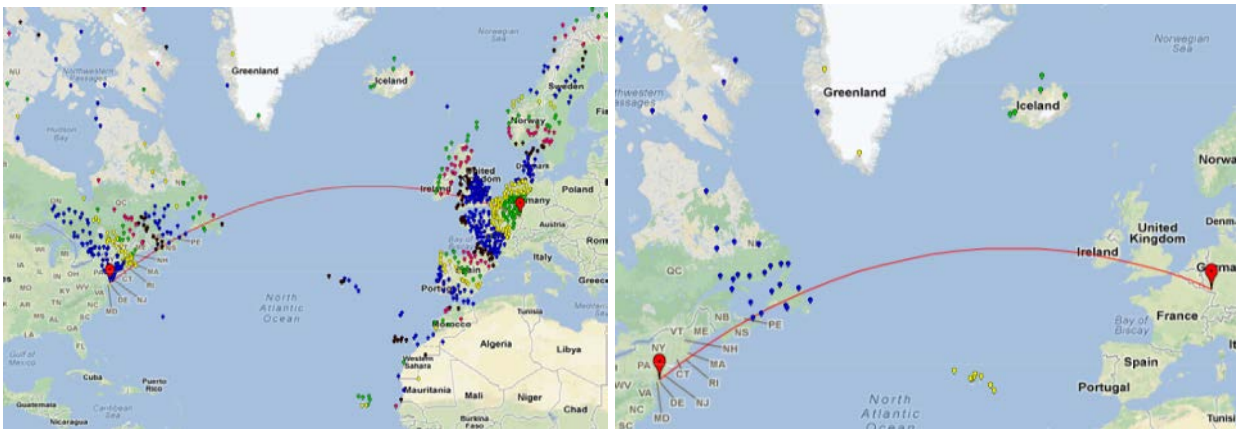
When comparing an origin-destination flight to an origin-en route stop-destination flight, is there some minimum distance below which it is more valuable to fly direct to the destination? If such a minimum distance exists, then perhaps that distance could be removed from the "Lens" as established by Equations 15 and 16. This would reduce the number of nodes for path analysis without negatively impacting solution quality. This distance will be referred to as the minimum-cutoff distance.

$$\delta_{OE} < \delta_{OD} - \gamma \quad (15)$$

$$\delta_{ED} < \delta_{OD} - \gamma \quad (16)$$

Where γ = Minimum-Cutoff Distance

The inclusion of a minimum-cutoff distance has two impacts. First, it compresses the eye shape formed by the original constraint. Second, it limits computation time by setting a limit to the maximum number of en route stops. Figure 13 illustrates the impact of the application of the minimum-cutoff distance. The example shows a requirement from Dover airfield in Delaware to Ramstein airfield in Germany on a Google Maps flat earth projection. The color bands are distance bands from the destination. By including an arbitrary minimum-cutoff distance of 700 nautical miles (NMs), the number of potential airfields is reduced from 812 to 50.



No Minimum Cutoff Distance 700 NM Minimum Cutoff Distance

Figure 13: Impact of minimum cutoff distance on nodal reduction

This minimum-cutoff distance affects routing algorithms not only by limiting the number of nodes, but also by limiting the number of potential stops. To highlight the impact of minimum-cutoff distance using simple enumeration, the maximum number of legs in Figure 13 is reduced from 812 to 4 and the number of routing alternatives is reduced from 2^{812} to 241. This example suggests that a minimum-cutoff distance could have potential value for routing analysis. Although enumeration was used, any solution technique with a computation time based on the number of nodes would benefit from this research. For example, both Karmarkar's (1984) algorithm for linear programming, $O(n^{3.5})$, and Dijkstra's (1959) algorithm for shortest path, $O(n^2)$, would benefit from the nodal reduction methodology offered by this research.

To calculate this minimum-cutoff distance, a model needs to be established that associates aircraft flight distance with objective measures that the decision maker determines to be of value. This model was created using a multi-objective decision analysis technique similar to Brooks and Kirkwood (1988). This technique was based off of the value functions of Dyer and Sarin (1979). Jackson et al. (1996) attempted to apply a similar technique using a value focused thinking approach to airlift and included payload weight, payload volume, range, response and multi-role as measures of airlift value. These measures were given distinct values to contrast selected program alternatives. Reiman et al. (2011) found several airlift metrics to be of value including a Fuel Efficiency Index and load factors by weight and volume. It is the original contribution of this research to propose an airlift material distance value model to reduce path analysis computation time.

Airlift Distance Value Model

Value focused thinking requires alternatives to compare. Keeney (1994) states, "alternatives are relevant only because they are a means to achieve values." The means to

achieve values in our value model are the distances flown between airfields. The distances selected for the alternatives are based on the Vincenty elliptical earth distance (Vincenty, 1975). These distances are selected in 100 NM increments for simplicity. All measures selected for the value model are objectively determined from the alternative distance. These measures are divided into two categories; *effectiveness* and *efficiency*. Effectiveness includes payload movement and safety. Payload movement is a measure of the weight of material moved in tons multiplied by the distance moved over a given day. Safety is a measure of the circadian rhythm shift for the aircrew and indicates the hours of deviation from a 24 hour takeoff to next day takeoff time. Efficiency is concerned with cycle completion and fuel efficiency. Cycle completion is the ability to airlift material to a trans-load point and return to the originating airfield. Fuel efficiency is the ratio of the weight of material airlifted in tons multiplied by the distance moved in a day over the amount of fuel consumed to achieve that material movement. Figure 14 details the planning distance value hierarchy.

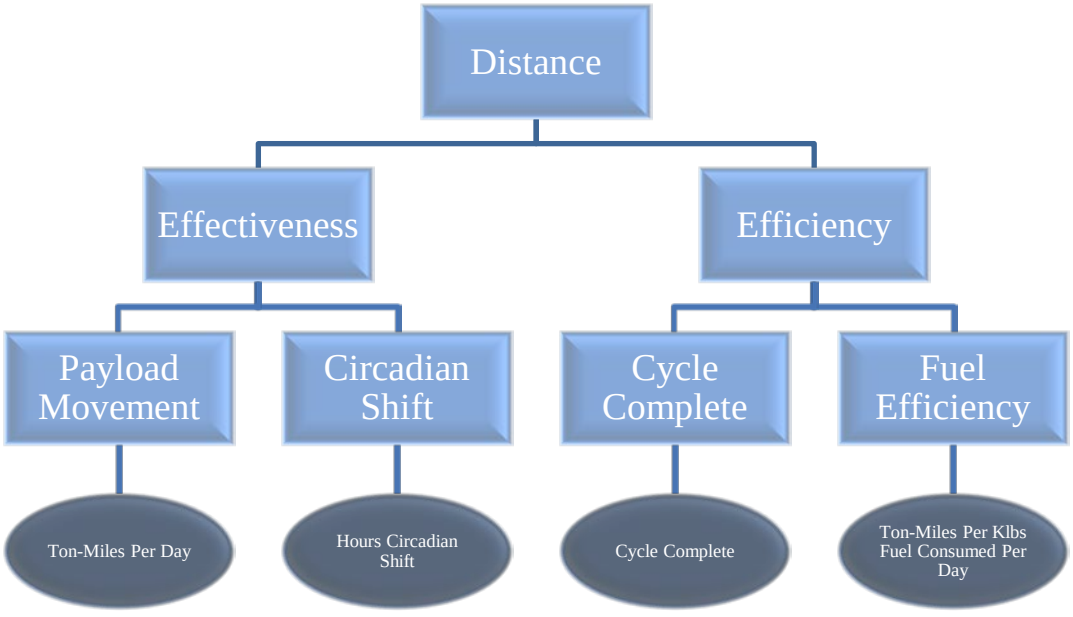


Figure 14: Strategic airlift problem planning distance value hierarchy

Payload Movement

Payload movement is an important measure of airlift effectiveness. The lack of payload movement in the 1991 Persian Gulf War resulted in the August 18, 1990 activation of the Civil Reserve Air Fleet (CRAF) (Thomchick, 1993). As Lewis (1998) points out, the civilian CRAF participants are in business to make money and if the business arrangement costs them money, then they would likely opt out of the CRAF program. Decreasing the negative impacts to CRAF partners and the economy during a major CRAF activation requires increasing military payload movement capability. Ton-miles was selected to measure this payload movement. Similar metrics for measuring payload movement are used by Lahiri et al. (2003), Owen (2008), and Hellermann et al. (2013). The distance in the ton-miles metric will be based on the Vincenty elliptical earth distance between on-load and off-load per Federal Code (CFR, 2010).

The amount of cargo flown a given distance is dependent upon many factors such as pavement strength, runway length, special departure procedure climb-out limitations, weather, flight altitude and winds. The variables that are based on environmental or airfield specific factors assume maximum cargo movement so as to be node agnostic. Table 1 provides values chosen for the key parameters in the payload calculation for three airlift aircraft. All weights in the table are measured in thousands of pounds (Klbs). A particular desired cruise altitude can be selected by the decision maker. We chose optimal cruise for *max gross takeoff weight* for our analysis.

Table 7: Aircraft-Specific Payload Movement Assumptions

	C-5B	C-17A	C-130J	References
Operating Weight	380	282.5	78	TO 1MDS-1
Max Gross Takeoff Weight	769	585	155	TO 1MDS-1
Fuel Capacity	347	241.36	62	TO 1MDS-1
Aircraft Max Payload	270	170.9	53	TO 1MDS-1
Reserve Fuel	23.45	21.44	4	AFI 11-2MDS V3
Alternate Fuel	23.45	21.44	4	AFI 11-2MDS V3
Holding Fuel	17.59	16.08	0	AFI 11-2MDS V3
Start Taxi Takeoff Fuel	3	4.5	0.67	TO 1MDS-1-1
Approach Fuel	7	2.67	0.7	TO 1MDS-1-1

To determine ton-miles, the maximum payload for a sortie over a given distance must be calculated. Sortie maximum payload weight is calculated according to Equation 17. Aircraft maximum payload weight, maximum gross takeoff weight and operating weight are assumed to be fixed. The weight of the ramp fuel is dependent upon the distance flown and payload weight as seen in Equation 18 (Yamani et al., 1990). Reserve, contingency, alternate and holding fuels are safety fuels that are not planned to be consumed. Start, taxi, takeoff and approach fuel are often planned to be the same for every sortie. Climb and descent fuel are based on planned cruise altitude and the aircraft gross weight at the start of takeoff and descent.

$$\omega_{pmax} = Min(\omega_{apmax}, (\omega_{mgt} - \omega_{rf} - \omega_{op})) \quad (17)$$

$$\omega_{rf} = \omega_{fstto} + \omega_{fc} + \omega_{ff} + \omega_{fd} + \omega_{fapp} + \omega_{frc} + \omega_{fah} \quad (18)$$

Where (All weights in Klbs):

ω_{pmax} = Sortie Maximum Payload Weight
ω_{apmax} = Aircraft Maximum Payload Weight
ω_{mgt} = Maximum Gross Takeoff Weight
ω_{rf} = Ramp Fuel Weight
ω_{op} = Operating Weight
ω_{fstto} = Start, Taxi & Takeoff Fuel Weight
ω_{fc} = Climb Fuel Weight
ω_{ff} = Cruise Fuel Weight
ω_{fd} = Descent Fuel Weight
ω_{fapp} = Approach Fuel Weight
ω_{frc} = Reserve/Contingency Fuel Weight
ω_{fah} = Alternate/Holding Fuel Weight

To determine the climb, cruise and descent fuels, regressions were performed on flight data from aircraft performance manuals. The climb regression equation is Equation 19 and the β parameters for Equation 19 are shown in Table 8. The lowest adjusted R^2 for any of the climb regressions was 0.9823. The descent regression equation is Equation 20 and the β parameters for Equation 20 are shown in Table 9. The lowest adjusted R^2 for any of the descent regressions was 0.9371.

$$\varphi_C = \beta_0 + \beta_1\alpha + \beta_2\alpha^2 + \beta_3\alpha^3 + \beta_4\omega + \beta_5\omega^2 + \beta_6\omega^3 + 10^{-6}\beta_7\alpha^2\omega^3 + 10^{-6}\beta_8\alpha^2\omega^3 \quad (19)$$

Where:

φ_C = Time to Climb in minutes, Fuel to Climb in Klbs or Distance to Climb in NMs
α = Altitude in Thousands of Feet
ω = Aircraft Gross Weight in Klbs at Climb Start

Table 8: Climb regression terms

	C-5 Climb φ_C			C-17 Climb φ_C			C-130 Climb φ_C		
	Time	Fuel	Dist	Time	Fuel	Dist	Time	Fuel	Dist
β_0	-9.2979	-3.0115	-44.558	-10.199	-4.7054	-51.504	-9.1135	-1.0670	-30.656
β_1	0.5454	0.3192	2.3817	0.5155	0.2869	2.0961	0.7454	0.0669	3.1655
β_2	-0.0197	-0.0082	-0.0861	-0.0136	-0.0070	-0.0282	-0.0313	-0.0022	-0.1600
β_3	0.0003	9.5E-05	0.0017	0.0002	7.1E-05	0.0003	0.0005	3.0E-05	0.0026
β_4	0.0451	0.0164	0.2454	0.0607	0.0267	0.3363	0.1654	0.0218	0.5290
β_5	-8.1E-05	-3.3E-05	-0.0005	-0.0001	-5.9E-05	-0.0008	-0.0014	-0.0002	-0.0050
β_6	4.9E-08	2.2E-08	2.9E-07	1.1E-07	4.8E-08	6.9E-07	4.2E-06	5.2E-07	1.7E-05
β_7	3.2E-05	3.7E-05	0.0001	8.0E-05	6.7E-05	0.0003	-0.0014	0.0003	-0.0107
β_8	1.4E-06	7.1E-08	1.1E-05	1.5E-06	-2.1E-07	1.7E-05	0.0003	1.3E-05	0.0014

$$\varphi_D = \beta_0 + \beta_1\omega + \beta_2\omega^2 + \beta_3\alpha + \beta_4\alpha\omega \quad (20)$$

Where:

φ_D = Time to Descend in minutes, Fuel to Descend in Klbs or Distance to Descend in NMs
ω = Aircraft Gross Weight in Klbs at Descent Start
α = Altitude in Thousands of Feet

Table 9: Descent regression terms

	C-5 Descent φ_D			C-17 Descent φ_D			C-130 Descent φ_D		
	Time	Fuel	Dist	Time	Fuel	Dist	Time	Fuel	Dist
β_0	-4.1137	-1.9673	-19.895	0.7301	0.2574	-16.382	-2.9838	-0.0513	-22.075
β_1	0.0186	0.0128	0.0767	0.0143	0.0005	0.1278	-0.0450	-0.0012	-0.0813
β_2	1.83E-5	1.34E-5	7.27E-5	-2.1E-5	-8.5E-7	-1.7E-4	0.0005	1.38E-5	0.0016
β_3	0.2282	0.1254	1.3771	0.2042	0.0108	1.3919	1.4471	0.0367	4.4627
β_4	0.0006	0.0004	0.0026	0.0006	3.2E-5	0.0036	-0.0056	-0.0002	-0.0143

The derivation of cruise fuel is similar to the technique used by Yamani et al. (1990). Yamani's linear regression of the specific range charts held altitude constant at 31,000 feet. To allow for more flexibility in altitude selection, we computed a new regression that allows both aircraft gross weight and altitude to be independent variables. The regression equation for specific range is in Equation 21. Table 10 shows the β terms for each aircraft type. The lowest adjusted R^2 for any of the specific range regressions was 0.9914.

$$\theta = \beta_0 + \beta_1\alpha + \beta_2\alpha^2 + \beta_3\omega + \beta_4\omega^2 + \beta_5\alpha\omega \quad (21)$$

Where:

θ = Specific Range in NMs per Klbs
α = Altitude in Thousands of Feet
ω = Aircraft Gross Weight in Klbs

Table 10: Specific range regression terms

	C-5	C-17	C-130
β_0	24.538	31.735	58.829
β_1	0.5511	0.9897	3.5292
β_2	0.0002	-0.0043	-0.0098
β_3	-0.0318	-0.0642	-0.2384
β_4	1.9E-05	5.8E-05	0.0010
β_5	-0.0005	-0.0011	-0.0155

Given the specific range regression equation, the distance flown in NMs for a given altitude and gross weight can be determined by integrating Equation 21 with respect to the change in fuel consumed over the interval from zero to the total fuel consumed as shown in Equation 22. After integrating and solving for cruise fuel, the resulting equation is as shown in Equation 23.

$$\delta = \int_0^{\omega_{ff}} (\beta_0 + \beta_1\alpha + \beta_2\alpha^2 + \beta_3\omega + \beta_4\omega^2 + \beta_5\alpha\omega)\, df \tag{22}$$

$$\omega_{ff} = -\frac{B}{3A} - \frac{1}{3A}\sqrt[3]{\frac{1}{2}\left[2B^3 - 9ABC + 27A^2D + \sqrt{(2B^3 - 9ABC + 27A^2D)^2 - 4(B^2 - 3AC)^3}\right]} - \frac{1}{3A}\sqrt[3]{\frac{1}{2}\left[2B^3 - 9ABC + 27A^2D - \sqrt{(2B^3 - 9ABC + 27A^2D)^2 - 4(B^2 - 3AC)^3}\right]} \tag{23}$$

Where (All weights in Klbs):

A $= \frac{\beta_4}{3}$

B $= \left(\frac{\beta_3}{2} + \beta_4(\omega_{op} + \omega_{frc} + \omega_{fah} + \omega_p) + \frac{\beta_5}{2}\alpha\right)$

C $= \beta_0 + \beta_1\alpha + \beta_2\alpha^2 + \beta_3(\omega_{op} + \omega_{frc} + \omega_{fah} + \omega_p) + \beta_4(\omega_{op} + \omega_{frc} + \omega_{fah} + \omega_p)^2 + \beta_5\alpha(\omega_{op} + \omega_{frc} + \omega_{fah} + \omega_p)$

D $= -\delta$

δ = Distance in NMs

α = Altitude in Thousands of Feet

ω = Aircraft Gross Weight
$= \omega_{op} + \omega_{frc} + \omega_{fah} + \omega_p + f$

ω_{op} = Operating Weight

ω_{frc} = Reserve/Contingency Fuel Weight

ω_{fah} = Alternate/Holding Fuel Weight

ω_p = Payload Weight

f = Fuel Consumed

ω_{ff} = Cruise Fuel Weight

The cruise fuel weight from Equation 23 is dependent upon the payload weight and the distance flown. To determine the maximum payload weight for a given distance, an iterative algorithm was utilized. The algorithm determines the ramp fuel using a given distance and zero payload. Using that ramp fuel, a new maximum payload is calculated according to Equation 17.

Using that payload a new ramp fuel is calculated. The process is iterated until the difference between the ramp fuel and the ramp fuel from the previous iteration falls below some threshold.

Using this method a maximum payload range curve can be created as shown in Figure 15. The first knee in the graph identifies the point at which payload is swapped for fuel. The primary measure of effectiveness, ton-miles, therefore rapidly increases before this point. The second knee in the graph is where payload is swapped for range and the fuel tanks are full. The effect of breaking up a flight distance into two segments is also apparent from the graph. For example, a C-17 travelling 4,000 NM can carry 64 Klbs of cargo, but segmenting that into two 2,000 NM sorties more than doubles cargo capacity to 143 Klbs. From a nodal reduction perspective, payload does not increase by adding an en route location when the distance is less than 700, 1,300 and 600 NM for the C-5, C-17 and C-130 respectively.

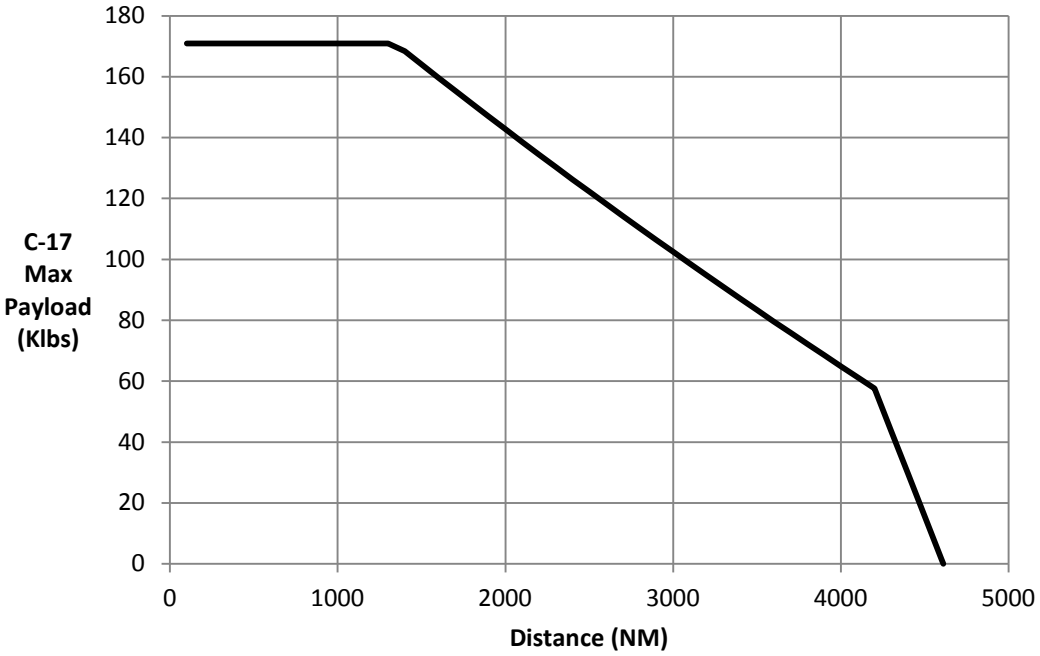


Figure 15: Maximum payload vs distance flown

To be able to determine the ton-mile per day capability over a given distance, it is important to understand the number of sorties that can be accomplished over a flight crew's allowable flying time. The number of sorties is calculated using Equation 24. The number of sorties assumes that if an aircraft and aircrew were capable of flying an additional sortie, then they would fly that sortie. To calculate the number of sorties it is necessary to calculate the time of flight for a given distance. The airspeed flown to obtain the specific range of Equation 21 is based off of regression Equation 25 for Mach airspeed, regression Equation 26 for true airspeed and the regression terms of Table 11. The lowest adjusted R^2 for any of the airspeed regressions was 0.9743.

$$\eta = Floor\left[\frac{\tau_{all}-\tau_{sto}}{\tau_f+\tau_g}\right] + \left\{ \begin{array}{ll} 1, & \tau_f \leq \tau_{all} - \tau_{sto} - Floor\left[\frac{\tau_{all}-\tau_{sto}}{\tau_f+\tau_g}\right](\tau_f+\tau_g) \\ 0, & \tau_f > \tau_{all} - \tau_{sto} - Floor\left[\frac{\tau_{all}-\tau_{sto}}{\tau_f+\tau_g}\right](\tau_f+\tau_g) \end{array} \right\} \quad (24)$$

$$\psi = \beta_0 + \beta_1\alpha + \beta_2\omega \quad (25)$$

$$\rho = \beta_0 + \beta_1\alpha + \beta_2\psi + \beta_3\alpha\psi \quad (26)$$

Where:

η = Number of Sorties
τ_{all} = Crew Show to Final Landing Time in Hours (16)
τ_{sto} = Crew Show at Airfield to Takeoff Time in Hours
(C-5: 4.25, C-17: 2.75, C-130: 2.25)
τ_g = Ground Time in Hours (C-5: 4.25, C-17: 3.25, C-130: 2.25)
τ_f = Takeoff-Landing Flight Time in Hours
= δ / ρ
δ = Distance in NMs
ρ = Flight True Air Speed in NMs Per Hour
ψ = Mach Airspeed
α = Altitude in Thousands of Feet
ω = Gross Weight of Aircraft in Klbs (Half of fuel consumed)

Table 11: Mach and True airspeed regression terms

	ψ			ρ
	C-5	C-17	C-130	
β_0	0.1683	0.2463	0.2209	0.7400
β_1	0.0101	0.0091	0.0055	-0.1302
β_2	0.0004	0.0004	0.0010	661.13
β_3				-2.2880

Ton-miles per day is based on the maximum payload from Equation 17 and the number of sorties from Equation 24. Two issues arise during the calculation of ton-miles. First, comparing the ton-mile outputs of an aircraft that has flown to the destination to another aircraft that has flown to the destination and back over the regulation mandated maximum flight time is misleading. Both outputs would be the same over one flight period, but combining the outputs over two flight periods would show a distinct doubling of output for the aircraft that can proceed to destination and return. To address this situation, both ton-mile outputs are calculated over two flight periods and then divided by two.

The second issue is that there is an output benefit to finishing the mission before the regulation maximum flight period is reached. This output benefit is dependent on airfield operating hours. To simplify analysis, we assume that airfield operating hours are not a limiting factor. To address the second issue, an adjustment is made that multiplies the output by the ratio of the number of hours in one day to the number of hours from the initial takeoff to the next possible takeoff after the crew has been able to rest. The calculation for the time from takeoff to next day takeoff is shown in Equation 27. Equation 28 shows the calculation of the ton-mile per day metric with both adjustments.

$$\tau_{toto} = \tau_f \eta + \tau_g(\eta - 1) + \tau_{crto} \tag{27}$$

$$\mu_{tmpd} = \frac{6\omega_{pmax}\delta\eta}{\tau_{toto}} \tag{28}$$

Where:

τ_{toto} = Takeoff to Next Takeoff after Crew Rest Time in Hours
τ_f = Takeoff-Landing Flight Time in Hours
η = Number of Sorties
τ_g = Ground Time in Hours
τ_{crto} = Aircrew Entering Crew Rest to Takeoff Time in Hours
(C-5: 17, C-17: 16.5, C-130: 16)
μ_{tmpd} = Ton-Miles Per Day
ω_{pmax} = Sortie Maximum Payload Weight in Klbs
δ = Distance in NMs
τ_{toto} = Takeoff to Next Takeoff after Crew Rest Time in Hours

Value focused thinking theory uses single dimensional value functions to translate preferentially independent measures to a common scale of zero to one. The value functions for this analysis were modeled linearly with maximum values representing the maximum possible for the respective aircraft type. The value function for the ton mile metric is shown in Equation 29.

$$\upsilon_{tmpd} = \begin{cases} \frac{\mu_{tmpd}}{\mu_{tmpd\,max}}, & \mu_{tmpd} \leq \mu_{tmpd\,max} \\ 1, & \mu_{tmpd} > \mu_{tmpd\,max} \end{cases} \tag{29}$$

Where:

υ_{tmpd} = Value of Ton-Miles Per Day on Scale from Zero to One
μ_{tmpd} = Ton-Miles Per Day
$\mu_{tmpd\,max}$ = Maximum Ton-Miles Per Day
(C-5: 130,630, C-17: 118,400, C-130: 34,860)

Cycle Complete

A cycle is defined as "an aircraft movement through a route from an origin, or Aerial Port of Embarkation (APOE), through en routes where the aircraft may receive additional cargo, passengers, fuel, minor maintenance, and a different crew, to its destination, or Aerial Port of

Debarkation (APOD), where it is unloaded and returns to its origin" (Watson, 2003). Cycle completion refers to the ability of an aircraft to depart and return to its original location. Assuming that the departure location is a staging location where aircrews, aircraft maintenance and mission handling equipment infrastructure are available, then there is potential value to the decision maker for the aircraft to depart and return to the starting location. If aircrews are able to reside at their dwellings then organizations can save on lodging expenses and reduce aircrew-family disruption. In addition, additional aircraft could be used in case of maintenance issues. Since fewer inputs are required to achieve a greater output if a cycle is complete, cycle completion is viewed as a measure of efficiency.

A critical assumption of the value of cycle complete is that the aircraft will trans-load cargo at the en route location. If trans-load operations are not planned, then the decision maker should weight cycle complete zero. If trans-load operations are planned, then proper personnel and mission handling equipment must be available at the trans-load airfield. Murphy's et al. (2008) survey of Air Cargo companies ranks equipment availability at trans-load locations as the top concern. To determine whether an aircraft is cycle complete depends upon the flight distance involved and the regulation maximum aircrew flying hours. The Federal Aviation Administration limits flight duty to 16 hours per 24 hour period (FAA, 1996). We calculate cycle completion based on the number of sorties in Equation 24. If the number is even then the value for cycle complete is one, else the value is zero.

Fuel Efficiency

Utilizing the ton-miles per day from Equation 28 and dividing by the total fuel consumed from Equation 30 adjusted for fuel consumed per day will result in the ton-miles per Klbs of fuel consumed metric of Equation 31. This ratio of payload movement output to fuel consumption

input is a measure of efficiency. Improving this ratio is of potential value to the decision maker, since it results in both a cost reduction for a given level of output and a smaller logistics footprint. Equation 30 is based off the underlying assumption that the aircraft will airlift maximum payload on all sorties. This assumption simplifies the analysis. The value function for this metric is modeled similar to the ton-mile per day metric with maximum values representing the maximum possible for the respective aircraft type. The value function for the ton-mile per Klbs of fuel consumed per day metric is shown in Equation 32.

$$\omega_{fcons} = \eta\left(\omega_{stto} + \omega_{fc} + \omega_{ff} + \omega_{fd} + \omega_{fapp}\right) \tag{30}$$

$$\mu_{tmpkpd} = \frac{\mu_{tmpd}\tau_{toto}}{24\omega_{fcons}} \tag{31}$$

$$\upsilon_{tmpkpd} = \begin{cases} \frac{\mu_{tmpkpd}}{\mu_{tmpkpd\,max}}, & \mu_{tmpkpd} \leq \mu_{tmpkpd\,max} \\ 1, & \mu_{tmpkpd} > \mu_{tmpkpd\,max} \end{cases} \tag{32}$$

Where (All weights in Klbs):

ω_{fcons} = Total Fuel Consumed During Sortie
η = Number of Sorties
ω_{fstto} = Start, Taxi and Takeoff Fuel Weight
ω_{fc} = Climb Fuel Weight
ω_{ff} = Cruise Fuel Weight
ω_{fd} = Descent Fuel Weight
ω_{fapp} = Approach Fuel Weight
μ_{tmpkpd} = Ton-Miles Per Klbs of Fuel Consumed Per Day
τ_{toto} = Takeoff to Next Takeoff after Crew Rest Time in Hours
υ_{tmpkpd} = Value of Ton-Miles Per Klbs of Fuel Consumed Per Day
$\mu_{tmpkpd\,max}$ = Maximum Ton-Miles Per Klbs Per Day
(C-5: 747, C-17: 798, C-130: 865)

Circadian Rhythm

Circadian rhythm is a human factor consideration. Aircrew fatigue is negatively impacted by time shifting operations from the current circadian rhythm of the aircrew (Samm and Perelli, 1982). Crum and Morrow (2002) conclude that fatigue can be reduced by providing the opportunity to sleep during normal sleeping hours. Increasing aircrew fatigue can negatively

impact safety of a flight and increase risk. Jackson et al. (1999) highlighted risk as an important factor when selecting among alternatives in their VFT model. Regulations are established that address crew fatigue issues. These regulations limit the maximum flight duration for a given crew complement (USAF, 2011). They address fatigue from flight duration, but do not address the negative impacts of cascading circadian rhythm. Selection of en route locations for trans-load operations can alter circadian rhythm cycles for enhanced safety. Keeping a stable circadian rhythm also has ancillary benefits of greater aircrew utilization, more stable planning and routine operations.

Circadian rhythm is measured from the amount of time that the aircrew's circadian clock gets shifted. If the time from initial takeoff to next day's takeoff is less than or equal to 24 hours, then the optimal value of one will be received. As the time shifts, value decreases until the time from initial takeoff to next day's takeoff equals the allowable flight period plus crew rest. The time for circadian shift is calculated by subtracting twenty four hours from the time from initial takeoff to next takeoff after crew rest as seen in Equation 33. This time from initial takeoff to next day takeoff is the same as calculated in Equation 27. The value function for the circadian shift time is linear over the range of zero to the maximum possible for the respective aircraft type as shown in Equation 34.

$$\mu_{cs} = \tau_{toto} - 24 \tag{33}$$

$$\upsilon_{cs} = \begin{cases} \frac{\mu_{cs\,max} - \mu_{cs}}{\mu_{cs\,max}}, & \mu_{cs} > 0 \\ 1, & \mu_{cs} \leq 0 \end{cases} \tag{34}$$

Where:

μ_{cs} = Circadian Shift in Hours
τ_{toto} = Takeoff to Next Takeoff after Crew Rest Time in Hours
υ_{cs} = Value of Circadian Shift on Scale from Zero to One
$\mu_{cs\,max}$ = Maximum Circadian Shift in Hours
(C-5: 4.75, C-17: 5.75, C-130: 5.75)

Material Airlift Distance Value Model Weights

The strategic decision maker must assign weights for each of the four metrics so that they sum to one. We will examine three scenarios for analysis. Scenario one is that the decision maker weights payload movement a value of one and everything else zero. Scenario two is that the decision maker weights fuel efficiency a value of one and everything else zero. Scenario three is that the decision maker weights payload movement at 0.6, fuel efficiency at 0.2, circadian shift at 0.15 and cycle complete at 0.05. To calculate the value associated with a given distance, each measure's value will be multiplied by its weighting and summed according to Equation 35.

$$\upsilon_{\delta} = w_{cc}\upsilon_{cc} + w_{tmpd}\upsilon_{tmpd} + w_{tmpdkpd}\upsilon_{tmpkpd} + w_{cs}\upsilon_{cs} \tag{35}$$

Where:

υ_{δ} = Distance Value
w_{cc} = Weighting on Cycle Complete
υ_{cc} = Value of Cycle Complete
w_{tmpd} = Weighting on Ton-Miles Per Day
υ_{tmpd} = Value of Ton-Miles Per Day
w_{tmpkpd} = Weighting on Ton-Miles Per Klbs Fuel Consumed Per Day
υ_{tmpkpd} = Value of Ton-Miles Per Klbs Fuel Consumed Per Day
w_{cs} = Weighting on Circadian Shift
υ_{cs} = Value of Circadian Shift

Cutoff Distance Model

Two questions are important during route planning for the strategic airlift of requirements. First, should an en route location be selected between on-load and off-load? Second, if an airfield is to be selected then what distance ranges should be examined to identify optimal alternatives? For the first question, the benefits to stopping at an en route location include "cost to carry" avoidance, cargo transfer to maximize each sorties available payload, and increased payload potential due to reduced fuel load. "Cost to carry" refers to the amount of fuel

that is burned due to the added weight of the fuel being carried. The negatives to stopping at an en route location include increased fuel use for additional climb-out and airfield approach, decreased capacity utilization due to ground time, increased need for ground support, and increased risk for aircraft maintenance. The fuel required for the additional climb-out and approach can often exceed the benefit from the reduced cost to carry. The probability for over-flight being more fuel efficient than stopping is higher if the en route location is not located exactly along the route of flight.

To make the en route stop of enough value to overcome the negatives requires the value proposition offered by increased payload. From the payload vs distance chart in Figure 15, it was shown that no payload increase is possible for sorties less than 1,200 NM for the C-17. Beyond those distances every 100 NM reduction is worth 4,000 pounds of cargo. For long distance sorties with light payloads, the impact of an en route stop is disproportionately greater. For example, separating a 2,000 and 4,000 NM C-17 sortie into two segments increases payload potential 30,000 pounds and 80,000 pounds respectively.

To determine whether an en route stop is warranted, the value associated with a given distance was contrasted against the average of the values of potential leg combinations. For example, if the requirement distance was 1,000 NMs, then the average of the 100 NM leg and the 900 NM leg values was contrasted against the value of 1,000 NM. If the value of the average of the legs failed to exceed the value of the requirement distance, then the next leg combination of 200 NM and 800NM would be examined. The first distance where the combination value exceeds the direct flight will be 100 NM above the minimum-cutoff distance for that requirement distance. This is repeated for each 100 NM increment to obtain a minimum-cutoff distance model for each aircraft type.

Results

The three scenarios for the value models illustrate the inverted U shape of value with respect to distance. Figure 16 shows the value associated with a complete weighting on payload movement for scenario one. The graph shows an inverted U for each aircraft type with peaks in value ranging from 1,400 NMs to 2,100 NMs. The graph suggests that when building a route from the origin through a series of en route stops to the destination, the route should not include sorties consisting of short distances since those would result in low value routing alternatives.

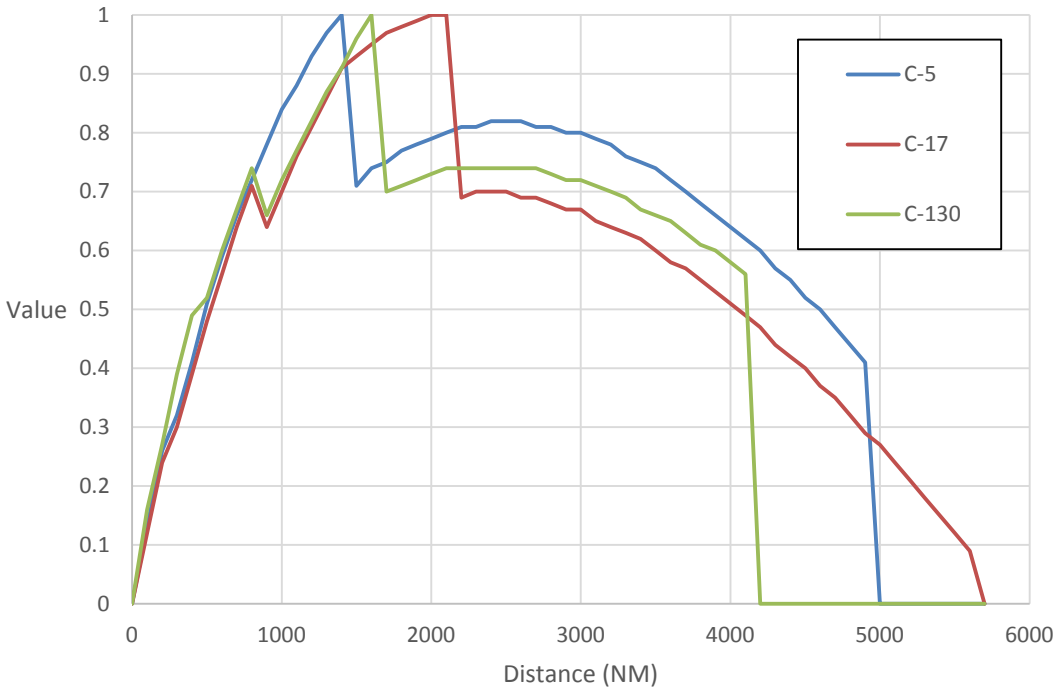


Figure 16: Scenario one value

The second scenario shows the value associated with fuel efficiency as can be seen in Figure 17. Once again the shape of the value curve is an inverted U. Short distances and long distances result in low value with optimal value being achieved in the range from 900 NM to 1,800 NM. The value associated with fuel efficiency tends to rise faster than payload movement. If an airlift planner is attempting to optimize on fuel efficiency, then the number of en route stops

is likely greater than the optimal number of stops for cargo throughput. In an anlysis of seventeen randomly selected origin and destination pairs, the number of en routes stops was sixteen percent greater when optimizing on fuel efficiency than when optimizing on cargo throughput.

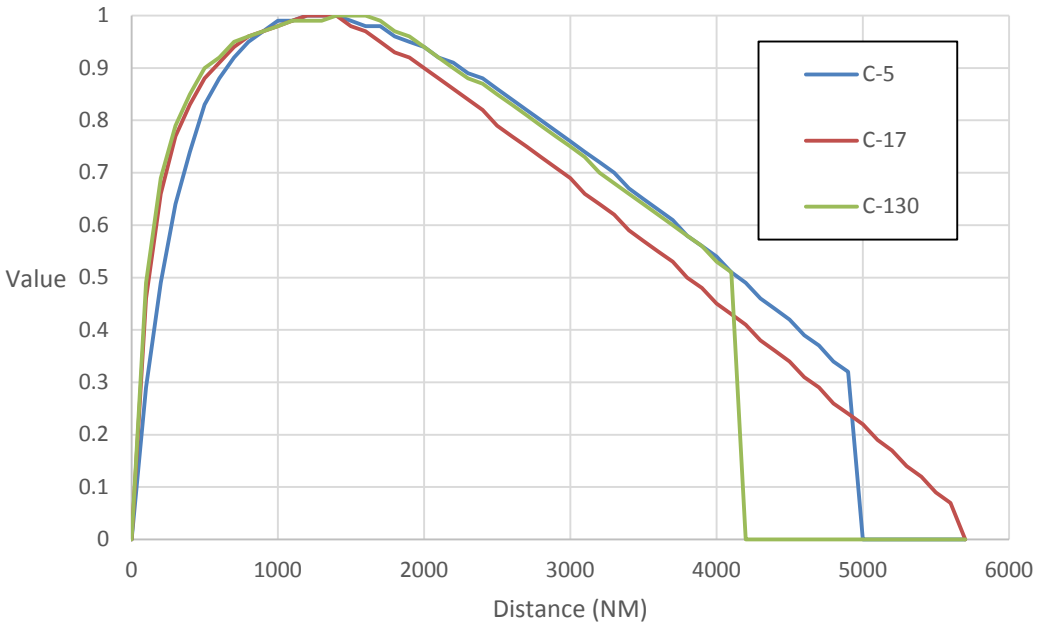


Figure 17: Scenario two value

The third scenario illustrates a mix of the four measures. The value graph for the third scenario is shown in Figure 18. The inverted U is apparent once again and the optimal range is from 1,300 NMs to 1,600 NMs. Three scenarios were selected to show the robustness of the inverted U to changes in subjective decision maker values. As long as there is a substantial weighting on payload movement and fuel efficiency, then short sortie distances and long sortie distances will tend to have low value.

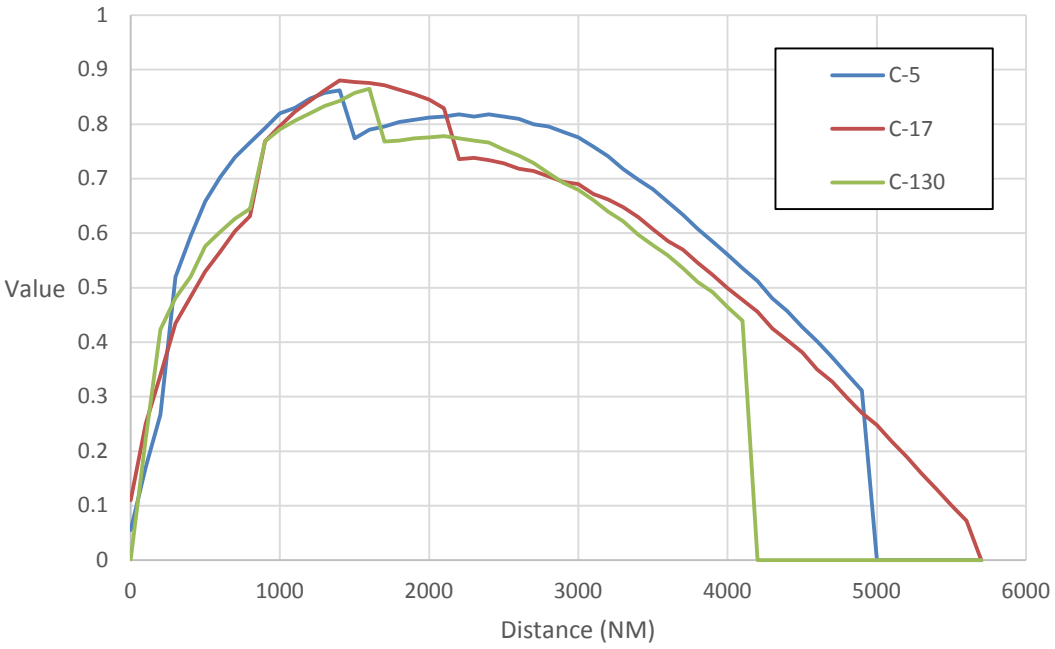


Figure 18: Scenario three value

Given these value models, we created a cutoff distance model that provides the minimum-cutoff distance given an origin and destination requirement distance. The results of the cutoff distance models for each scenario and aircraft type can be seen in Figure 19. In addition to the cutoff distance for each aircraft type a line is included that shows the minimum of the minimum-cutoff distances. Two important airlift planning implications arise from the three graphs. First, it is unnecessary to plan an en route stop if the requirement distance is less than 1,700 NMs. This observation eliminates the necessity for routing algorithms for any requirement under 1,700 NMs. Second, nodal reduction declines as distance increases.

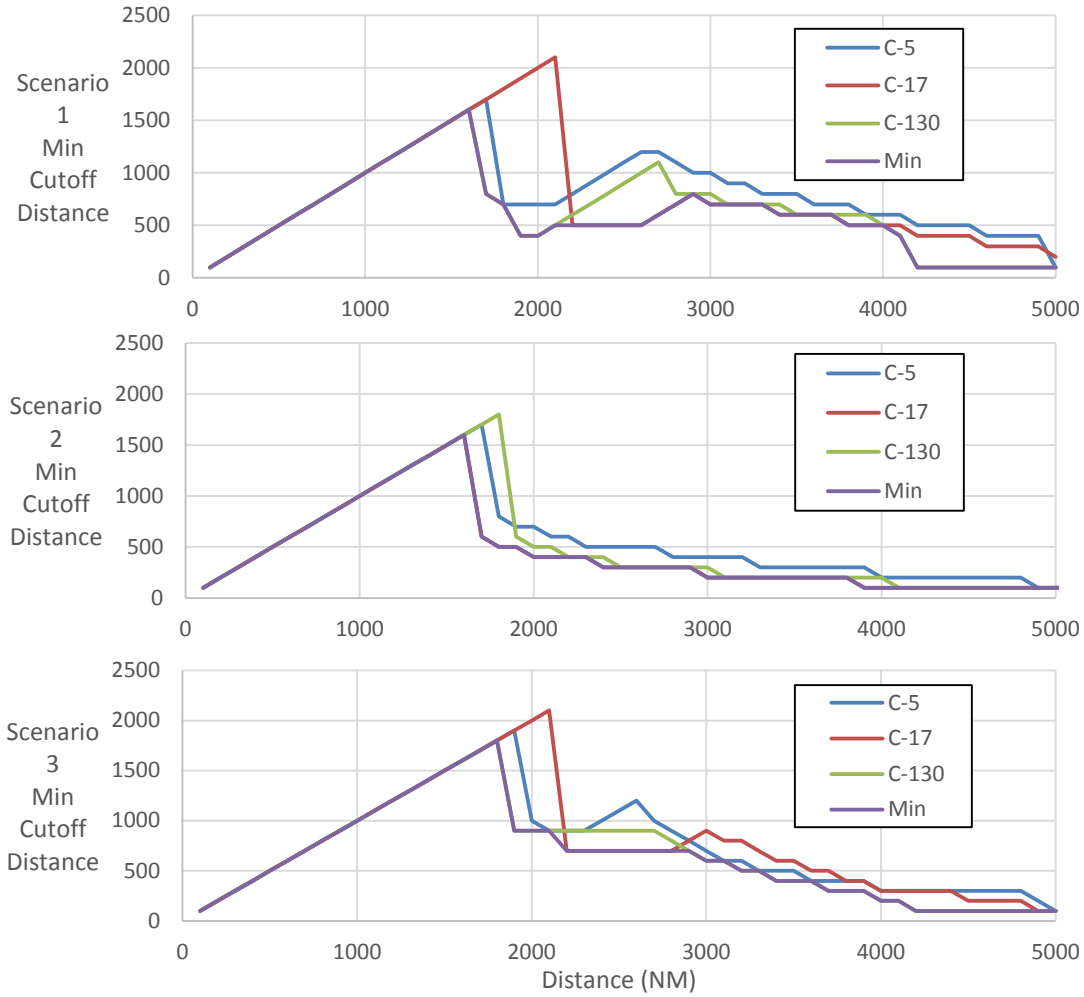


Figure 19: Cutoff Distance Models

To assess the impact of minimum-cutoff distance on path algorithm accuracy and speed, 50 random origin destination requirement pairs were selected. The optimal path was calculated for each requirement based off of the maximum cargo throughput in cargo Klbs per day and the maximum fuel efficiency in cargo Klbs per day per Klbs of fuel consumed. For simplicity, the path algorithm chosen was simple enumeration of all possibilities for up to three sortie paths. The cutoff distance model selected was based on the minimum for all three scenarios for the C-17 and is shown in Figure 20.

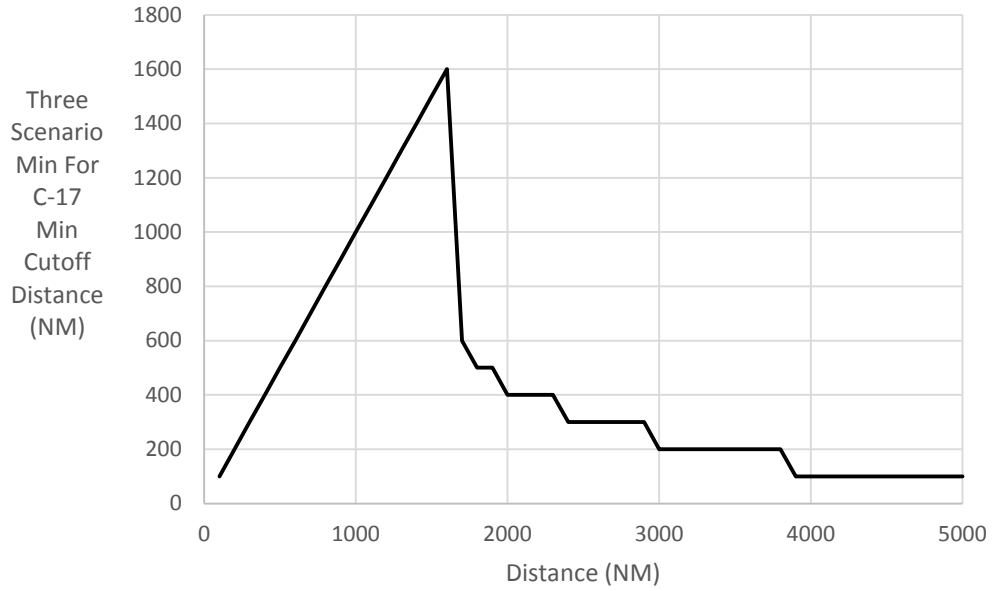


Figure 20: Minimum cutoff distance model used for analysis

In all of the fifty origin and destination pairs randomly selected, the optimal path for cargo throughput with nodal reduction was the same as the optimal path for cargo throughput without nodal reduction. The same is true for all fifty pairs with respect to fuel efficiency. For example, take the median distance sample point of Kaolack airfield, Senegal to Wilmington airfield in the United States. The optimal path for cargo throughput in both the "with nodal reduction" and "without nodal reduction" cases includes a single stop at Pico Islands airfield in the Azores and achieved a cargo throughput of 54 Klbs of cargo per day. The optimal path for fuel efficiency in both the "with nodal reduction" and "without nodal reduction" cases includes two stops. The first stop is the Santa Maria airfield in the Azores and the next stop is St. John's airfield in Newfoundland. The fuel efficiency for that route is 0.37 Klbs per day of cargo per Klbs of fuel consumed. Without nodal reduction, this origin-destination pair had 673 airfields in the lens, 83,812 route alternatives and took 41 seconds to calculate. With nodal reduction, this

origin-destination pair had 358 airfields in the lens, 11,095 route alternatives and took 12 seconds to calculate.

Examination of all fifty origin-destination pairs results in an average nodal reduction of 40%, an average route alternative reduction of 70% and an average computation time reduction of 55%. The reduction tended to be greater at lower distances as would be expected by the cutoff distance model. Figure 21 shows the percentage reduction in computation time by using nodal reduction for each of the 50 origin-destination pairs. There was one anomaly that required a portion of the airfields removed by the minimum-cutoff distance model to be reintroduced. This was due to pavement strength at the source airfield. The problem is that airfields with extremely low pavement strength will severely limit either fuel or payload. An aircraft at such an airfield could take a large payload and a small quantity of fuel, fly a short distance to a high pavement strength airfield, and fill up with fuel to fly the distance required. This is often the optimal solution for low pavement strength airfields for both cargo throughput and fuel efficiency.

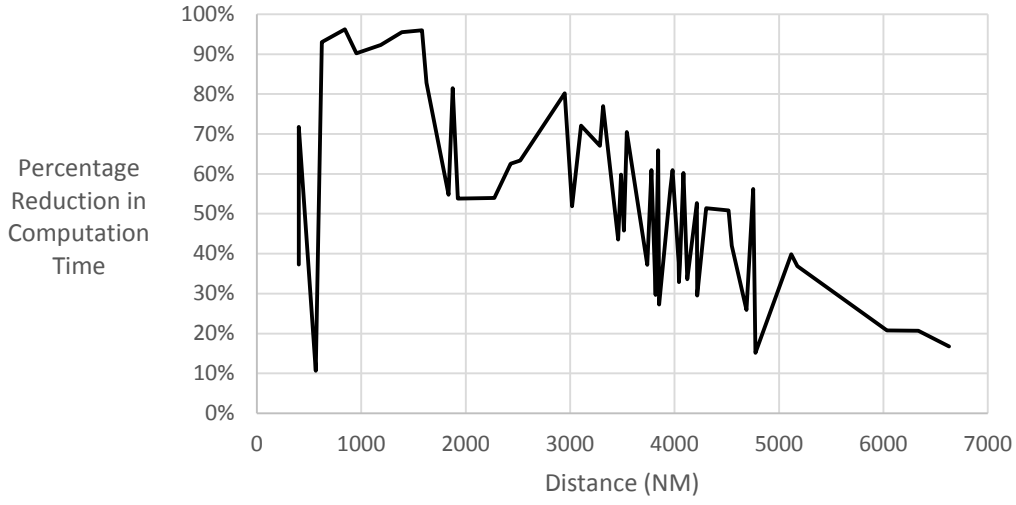


Figure 21: Nodal reduction computation time reduction

To resolve the pavement strength anomaly, when airfield pavement strength at the origin results in less than 90% maximum gross takeoff weight, then airfields previously removed that have a pavement strength that is at least 10% stronger than the origin airfield and are at distances from the destination that are between the distance from origin to destination and the distance from origin to destination minus minimum-cutoff are reintroduced. This can be seen by the small time reduction of 11% for the pavement restricted origin airfield at Niagara Falls in Figure 21. Of the 50 origin-destination pairs, 7 were affected by the pavement anomaly. It is worth mentioning that the pavement anomaly is also a situation when it would make sense to go backward away from the destination, so some thought might be given to adding in high pavement strength airfields close to the origin but farther from the destination than the source is to the destination.

Conclusion

The SAP requires the development of paths based from user requirements. A method to remove suboptimal airfield nodes from path generation would enhance the speed with which potential alternative paths can be processed and analyzed. Rapid path development for a heterogeneous set of aircraft is required prior to assigning cargo. Paths determine payload capability and payload capability determines what requirements should be assigned to which aircraft within an enterprise. How can the capacity of an edge of the capacitated vehicle routing problem be assigned without first assessing what the optimal path is for that edge? This research aids in the rapid calculation of this critical first step.

The strategic planning distance value model can be used for suboptimal airfield node removal, routing prioritization amongst alternatives, requirement aggregation and assignment of aircraft to requirements. Suboptimal airfield node removal is achieved through the minimum-

cutoff distance model. Speed is critical to airlift planning. External stimuli affecting one mission can quickly cascade to impact multiple missions. The ability to rapidly respond to these external stimuli can have a large impact on the overall efficiency of the enterprise. By providing the minimum-cutoff distance model, routing algorithm execution time is reduced. This in turn establishes capacity for the capacitated vehicle routing problem. Enabling operational airlift planners in both the civil aviation industry and the military to more rapidly optimize path alternatives through nodal reduction techniques has the potential to provide quicker analysis for time sensitive applications.

Nodal Reduction Heuristics Applied to Route Generation for Enterprise Airlift Evaluation

Introduction

The Strategic Airlift Problem (SAP) attempts to optimally schedule a set of aircraft to airlift a set of requirements over a set of airfields (Reiman et al., 2014). Rappoport et al. (1992) considered the airlift planning problem a vehicle routing problem with time and capacity constraints. Crino et al. (2004) described the SAP as a theater distribution vehicle routing and scheduling problem with the aircraft as modes and airfields as nodes. Given that the number of en route airfield nodes between requirement source and destination is *n* and assuming that each node visited is closer to the destination than the previous node, then the potential number of routing alternatives is 2^n. Balakrishnan suggests that routing decisions for the problem become complex because of the "large number of intermediate cities" (Balakrishnan, Chien, & Wong, 1989).

To understand the impact of this large number of intermediate cities, *n*, we selected the Digital Aeronautical Flight Information File (DAFIF) database of 5,342 unique International Civil Aviation Organization (ICAO) airfields. Without the assumption that each node has to be closer to the requirement destination than the last, approximately *e* multiplied by *n* factorial route combinations exist. This results in $e * 5{,}340!$ or $4.32 * 10^{17{,}588}$ potential route combinations. For a given requirement, examination of all of these alternatives would be too computationally extensive. Simple heuristics can rapidly reduce the number of routes without eliminating valuable alternatives. Bodin (1990) stated that, "because of the computational complexity in solving vehicle routing and scheduling problems to optimality, heuristics are employed."

Reiman et al. (2014) suggested two simple heuristics to reduce the number of route alternatives. The first heuristic suggests that there is no value flying to an en route location that is farther from the source than the source is to the destination. The second heuristic suggests that there is no value flying to an en route location that is farther from the destination than the source is to the destination. This results in the "eye shape" shown in Figure 22. Application of this heuristic to every en route node selection reduces the number of routes by reducing the size of n and by switching the route calculation formula from $e * n!$ to 2^n.

Figure 22: Eye shape (GCMap)

The impact of these simple heuristics on route combinations is highlighted in Table 12. Pascal's triangle (1665) can be seen in the route combinations of Table 12. The intersection of

each number of en route stops *k* and number of airfield nodes *n* in the table is calculated using Equation 36 and the "no heuristics" column is calculated using Equation 37. The problem highlighted by Table 12 is that when the selected requirement source and destination are near antipodal (opposite locations on the globe), *n* approaches 5,340 airfields. This results in too many alternatives to be rapidly analyzed. This indicates that further nodal reduction heuristics could be advantageous to reducing computation time.

$$\binom{n}{k} = \frac{n!}{(n-k)!(k!)} \tag{36}$$

$$\sum_{k=0}^{n} \frac{n!}{k!} \tag{37}$$

Table 12: Effect of increasing en route airfields on the number of routes

Number of En Route Airfields (*n*)	Number of En Route Stops (*k*)						Total 2^n	No Heuristics $\sum_{k=0}^{n} \frac{n!}{k!}$
	0	1	2	3	4	5		
0	1						1	1
1	1	1					2	2
2	1	2	1				4	5
3	1	3	3	1			8	16
4	1	4	6	4	1		16	65
5	1	5	10	10	5	1	32	326
6	1	6	15	20	15	6	64	1,957
7	1	7	21	35	35	21	128	13,700
8	1	8	28	56	70	56	256	109,601
9	1	9	36	84	126	126	512	986,410
10	1	10	45	120	210	252	1,024	9,864,101
5340	1	5340	1E+07	3E+10	3E+13	4E+16	3E+1,607	4E+17,588

We recommend several nodal reduction heuristics in an effort to reduce the number of potential routes analyzed. These include the minimum cutoff distance, total distance multiple, effective runway length, runway length, runway width, pavement strength, departure obstacles and diplomatic clearance heuristics. Minimum cutoff distance was developed by Reiman et al. (2014) and was derived from their distance value model. It attempts to remove airfield nodes that are located too close to the requirement source or destination. Total distance multiple represents the ratio of route length to the Vincenty elliptical Earth distance from source to destination (Vincenty, 1975). This heuristic suggests that airfields that are located farther from the great circle route often are of less value than those situated closer. Effective runway length is a heuristic that uses latitude, elevation and actual runway length to remove airfield nodes of low value. The runway length and width heuristics eliminate airfield nodes from consideration if their runway length or width is less than that mandated for a given aircraft type. Pavement strength filters remove unsuitable airfields based on the maximum aircraft gross takeoff weight that a given type of pavement can support. The departure obstacle heuristic seeks to remove airfields that require a high climb gradient which reduces the maximum aircraft gross takeoff weight. Finally, the diplomatic clearance heuristic reduces nodes by removing countries that pose diplomatic clearance difficulties. To understand the impact of these nodal reduction heuristics on route reduction, a sample set of requirements is used.

Requirements

A requirement is defined by cargo on-load at an origin airfield and cargo off-load at a destination airfield. Li et al. (2010) defined the origin and destination airfields as OD pairs. To establish a set of requirements, one hundred OD pairs are randomly selected from the 5,342 ICAO unique airfields in the DAFIF airfield database. The sample set of random OD pairs is

shown in Table 13. The distances between source and destination vary from 400 nautical miles (NMs) to over 10,000 NMs. The shortest distance OD pair is Gaborone, Botswana (FBSK) to Livingstone, Zambia (FLLI) and the longest pair is Barrow Island, Australia (YBWX) to Hacienda El Calvario, Venezuela (SVHD). The number of en route airfields within the eye-shape of each OD pair varies from 10 to 5,109 airfields.

Table 13: Randomly selected OD pairs

	From ICAO	To ICAO		From ICAO	To ICAO		From ICAO	To ICAO		From ICAO	To ICAO
1	FBSK	FLLI	26	GOOK	KILM	51	SBBT	LFRV	76	KNBC	ZSYN
2	OLRA	LTAC	27	CYUB	MHTG	52	FMCZ	EGQS	77	RPUR	KSAW
3	KIAG	CYQI	28	SBKG	LFMV	53	DNYO	KMMU	78	MUFL	OIKP
4	KLOZ	CYOW	29	OIKQ	FAUT	54	DABB	SKIB	79	OEJN	KBRO
5	KTUP	CYYB	30	PATE	EBBE	55	RJCB	LSMS	80	LTBG	YNTN
6	LEPP	DAUI	31	UHBB	PHIK	56	SBLO	LFRU	81	FATZ	KMCF
7	HHAS	ORSH	32	OIID	FAMM	57	CYYD	LGHI	82	PAGL	SBCH
8	UAOO	LTAU	33	LILH	TLPL	58	WBKS	PACX	83	PHJR	SBRF
9	TNCE	KFME	34	VTBD	YGLB	59	LFGF	MMGM	84	ESNJ	SAZN
10	MPHO	KGWO	35	MMBT	PAVL	60	LFMT	MMML	85	YWIS	SEAM
11	WITL	RPUT	36	CYAU	DGTK	61	KNUQ	ZBTJ	86	SCDW	VOCI
12	VOTP	OEAH	37	CZPC	EDAX	62	SETU	LIEO	87	EGNL	YKKG
13	KAJO	CYSN	38	WALG	UWOR	63	LRCL	KBAB	88	SVSA	RKRN
14	ZGOW	WATG	39	EPWR	KJYL	64	SEIN	LIPB	89	KABI	VDSV
15	FYTM	GLMR	40	SPTN	KSLC	65	EDCP	SPEP	90	SVOK	RKSM
16	VTUN	YMDY	41	KSGF	SBQV	66	KCPR	ROTM	91	YDPO	LKTB
17	LIBP	HCMH	42	DISP	ENSB	67	LFRD	WMKF	92	SKPQ	VOYK
18	CYVB	EDCD	43	RJSI	ESML	68	PCIS	MBAC	93	EGCC	YGDH
19	TJMZ	KMER	44	WIMN	LRIA	69	DNKT	PANT	94	SCVI	VARP
20	FOOK	EHRD	45	SLBJ	NTGT	70	YNBR	UTSB	95	FAMG	KTRM
21	CYPX	LHBC	46	SLPS	NTTO	71	TQPF	OYRN	96	NTKF	FWUU
22	CYTR	ETEK	47	FMNE	EGUO	72	KEGE	FKKM	97	EGPO	YMML
23	SAOR	MUSN	48	MMQT	EGCD	73	SAZP	EKVD	98	ETHF	NZKT
24	MTPP	LPMR	49	PANO	VYKG	74	YCMU	HAMK	99	ETNU	NZPM
25	SPHO	KPUB	50	DAUZ	SKSA	75	YJBY	PAFB	100	YBWX	SVHD

The specific source and destination can have a great impact on the number of airfields within the eye shape. Figure 23 illustrates the wide variance on the number of en route airfields associated with a given distance. Only one OD pair (FBSK-FLLI) out of those randomly selected results in less than one million routing alternatives. One option for reducing the number

of airfields in Figure 23 is to reduce the size of the eye shape from Figure 22. Two heuristic nodal reduction techniques lead to this reduction in size.

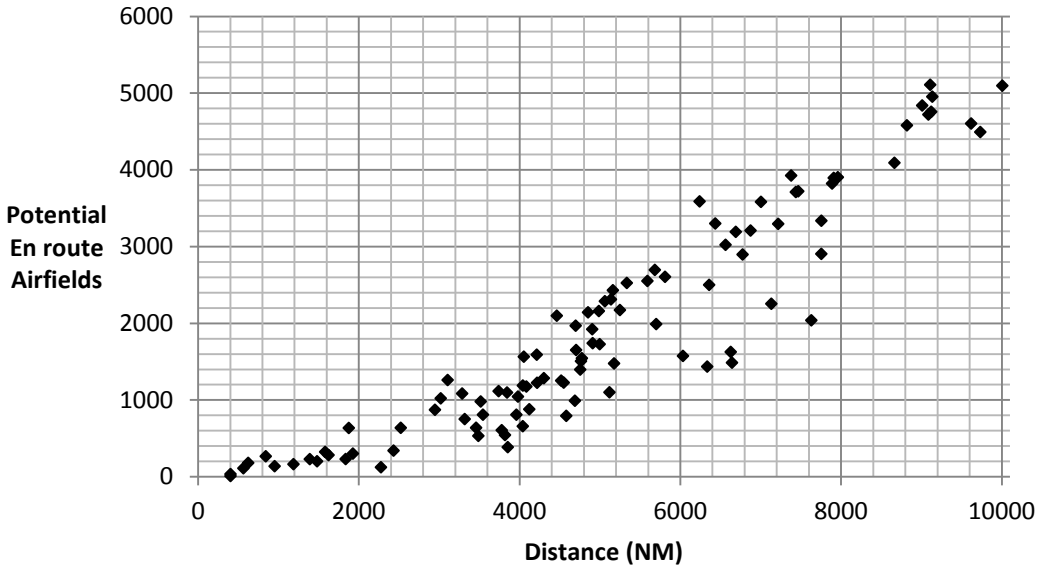


Figure 23: En route airfields vs distance for selected requirements

Requirement based heuristics

The two heuristic nodal reduction techniques for a given requirement are the minimum cutoff distance and the total distance multiple. The minimum cutoff distance heuristic reduces the size of the eye shape but does not alter the shape. The total distance multiple heuristic alters the eye shape so that the airfield nodes closest to the corners of the eye are eliminated first. We recommend using minimum cutoff distance before the total distance multiple since the total distance multiple can more easily select the exact number of airfields desired for analysis.

Minimum cutoff distance

Reiman et al. (2014) recommended a cutoff distance model where the addition of an airfield stop below a given distance would fail to add sufficient value to even consider that

option as an alternative. This would reduce the size of the eye shape as seen in Figure 24. The minimum cutoff distance serves two purposes. First, it reduces *n*, the number of airfields that can be found within the eye shape. Second, it limits *k*, the number of potential en route stops. The number of potential en route stops is limited by the floor of the ratio of the Vincenty elliptical earth distance between source and destination and the minimum cutoff distance minus one as seen in Equation 38. For example, for a 4,500 NM requirement, a 1,000 NM cutoff limits the number of *k* en route stops to three. Limiting the number of *k* has tremendous route reduction potential as can be seen in Table 12.

$$k = Floor\left(\frac{Vincenty\ Elliptical\ Earth\ Distance}{Minimum\ Cutoff\ Distance}\right) - 1 \tag{38}$$

Figure 24: Eye shape with cutoff distance applied (GC Mapper)

The cutoff distance model used is based off of the value model from Reiman et al. (2014). The value model weightings were changed to weight payload movement and fuel efficiency at 0.5. The exact cutoff distance model used is shown in Figure 25. The C-17 is range limited beyond 4,500 NMs. This does not allow for a value comparison to create the cutoff distance model beyond this distance. To adjust for this, we assume that airfield density is sufficient enough for OD pairs above 4,500 NM to have an airfield node available at least every 4,500 NMs. Given this assumption, all OD pairs beyond 4,500 NMs will use the cutoff of 400 NMs.

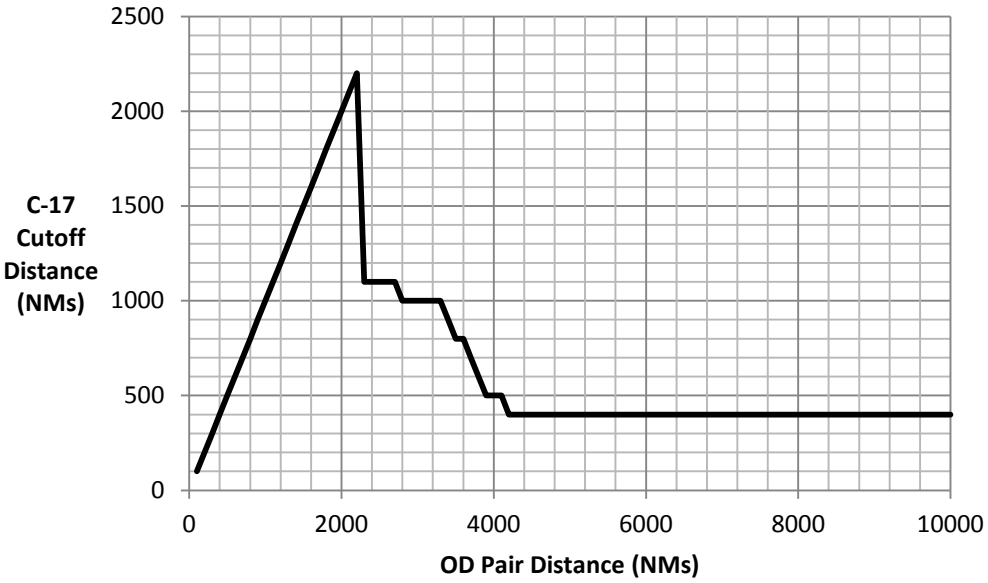


Figure 25: Cutoff distance model

Applying the minimum cutoff distance model to the 100 randomly selected OD pairs results in the airfield reductions highlighted in Figure 26. The use of the model leads to a 27% reduction in airfields. As OD pair distances increase, the potential for airfield reduction declines. At OD pair distances less than 4,000 NMs, nodal reduction is in excess of 70%. Although the

cutoff distance model performs well against requirements less than 4,000 NMs, there is a need for further airfield nodal reduction at distances greater than 4,000 NMs.

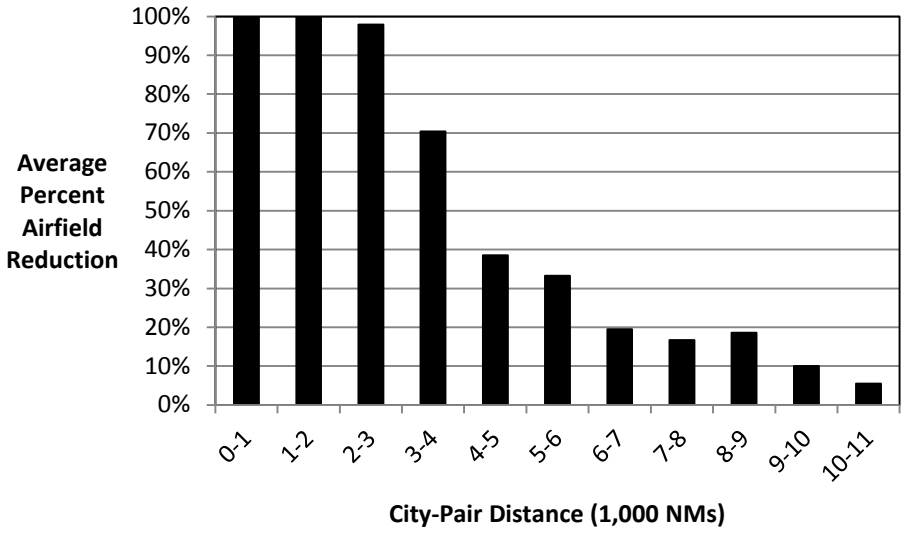


Figure 26: Average percentage airfield reduction vs OD pair distance

The minimum cutoff distance nodal reduction heuristic is applied iteratively for route creation. A route is defined as a set of sorties that create a path from origin to destination. A sortie is defined as an aircraft flight between two airfields with no en route stops in between. Minimum cutoff distance is applied to each sortie of a route. Application of the minimum cutoff distance to an OD pair will result in the set of potential airfields for one en route stop. This set of airfields is labeled primary en route airfields. After selection of a primary en route airfield as an en route stop, the primary en route airfield is treated similarly to the requirement source. A Vincenty elliptical Earth distance and a minimum cutoff distance are calculated for the new OD pair. A new eye shape is formed creating a set of secondary en route airfields associated with a

given primary airfield. Given the set of primary en route airfields and the set of secondary en route airfields associated with each primary, all potential routing alternatives can be calculated.

All primary en route airfields are cycled through for one stop routes. Then, each primary loops through all of their secondary airfields to create all two stop routes. For three stop routes, the secondary airfields will be matched against primary airfields. If the matched primary has secondary airfields, then all three stop routes will be created. This process will continue until all route combinations have been created. Given the high number of primary en route airfields at distances over 4,000 NMs, even with the utilization of the minimum cutoff distance heuristic, the number of route combinations is still too large for rapid computation and analysis.

Total distance multiple

Unlike the cutoff distance model, the total distance multiple provides for tailored nodal reduction at OD pair distances in excess of 4,000 NMs. Total distance multiple calculates the ratio of route length to minimum route length. The theory is that there is more value to airlifting over the minimum distance possible. Additional distance travelled in a no wind or constant wind field scenario results in more fuel consumed and often less payload available. To reduce fuel consumption and maximize payload, the goal is to minimize the total distance multiple. Total distance multiple is calculated according to Equation 39. For one en route stop, using the eye shape airfield constraint, the range of this ratio is limited from one to two. If the en route stop is along the Vincenty elliptical Earth path, the ratio is one and if the en route stop is at the corner of the eye, the ratio is two. Examples of limiting this ratio are shown in Figure 27 for the OD pair San Pedro, Cote D' Ivoire (DISP) to Longyear, Norway (ENSB). In the two and 1.5 distance multiples, airfields in North America are considered as options. This illustrates the weak value posed by airfields at high distance multiples.

$$\psi = \frac{\sum_{i=1}^{j} \delta_i}{\gamma} \tag{39}$$

Where:

ψ = Total distance multiple for en route airfield
i = Sortie number of the route
j = Total number of sorties on the route
δ_i = Vincenty elliptical Earth distance for that sortie
γ = Vincenty elliptical Earth distance for the OD pair

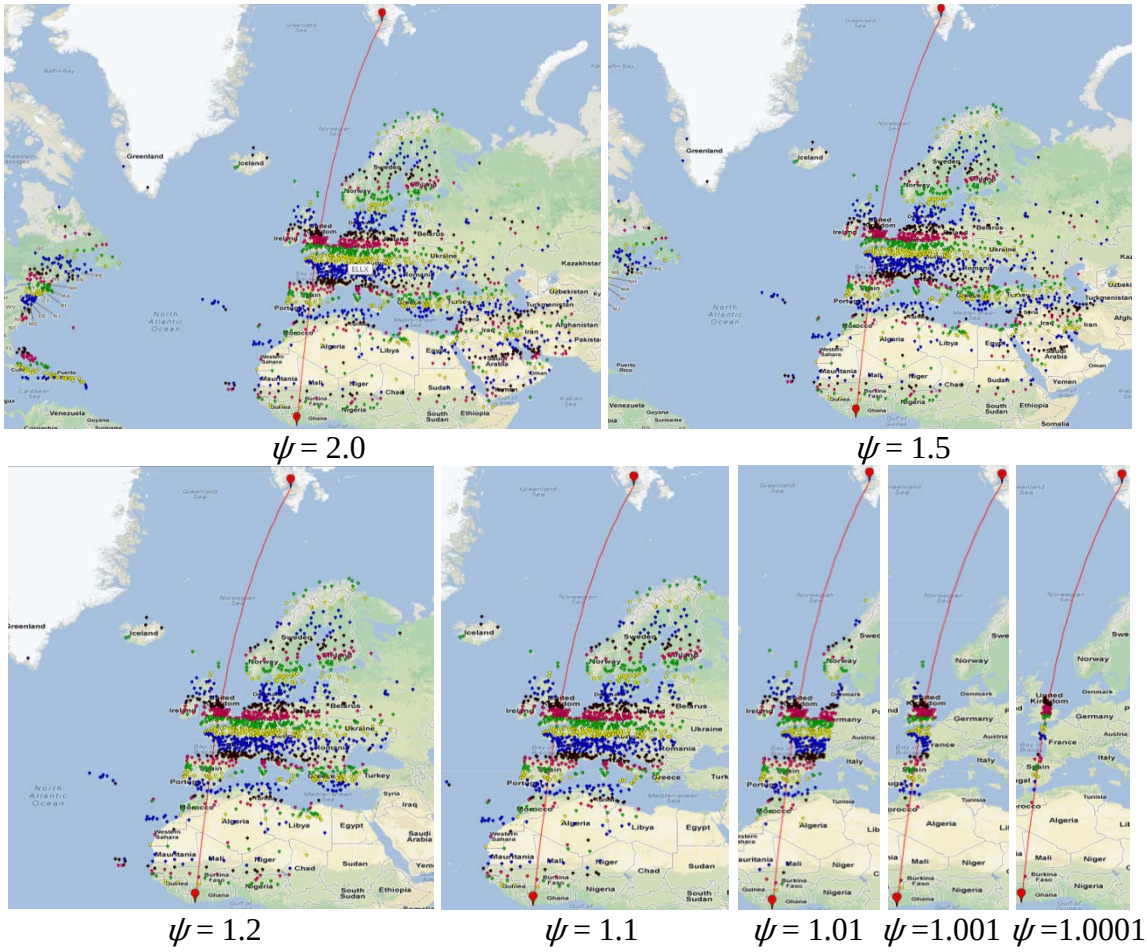


Figure 27: Airfield reduction by constraining ψ (Google Maps API)

If ψ is calculated for all routes with one en route stop, then airfields can be prioritized based on their ψ and only the top number of airfields selected by the decision maker can be

included for further analysis. For example, the decision maker can request only to analyze the top 500 airfields closest to the Vincenty elliptical Earth route. This empowers the decision maker to control the trade-off between the number of potential alternatives and the computation time required to give a result. To better illustrate the relationship between total distance multiple and value, 30 OD pairs are selected at evenly spaced intervals. Value is determined using the Reiman et al. (2014) value model. An example of the 25th, 50th, 75th, and 100th distance ranked OD pair's route value data are displayed in Figure 28. Each data point in the plot represents a potential en route airfield stop. The data shows that the highest airlift value is reduced as ψ increases. In all cases, eliminating distance multiples greater than 1.4 results in the removal of only airfields whose value was in the bottom 80% of value.

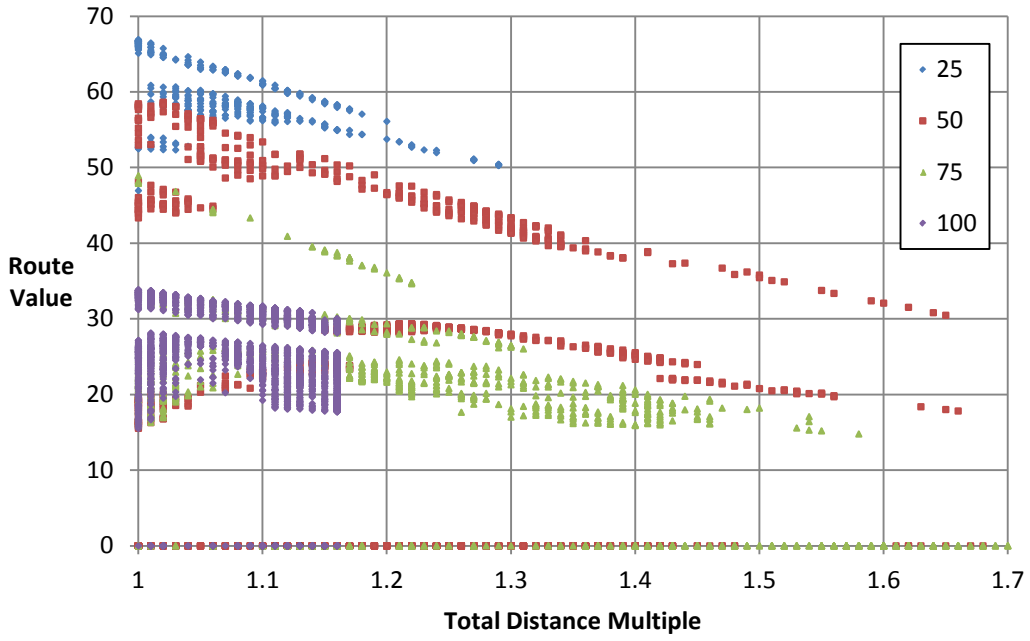


Figure 28: Impact of ψ on route value

The total distance multiple is used to select a set number of top ranked airfields for both the primary en route airfields and secondary en route airfields to establish the desired computation time and accuracy for route creation. The caveat is that setting the maximum number of primary or secondary airfields too low, might remove some valuable alternatives. Prior to route creation and establishment of the primary and secondary airfields, other filters to remove extremely low valued airfields should be utilized.

Airfield characteristic heuristics

What airfield characteristics should be used to eliminate low value route options? Baker et al. (2002) selected Maximum on the Ground (MOG) and fuel when modeling airfield constraints in their NPS/Rand Mobility Optimizer (NRMO). Yet, MOG and fuel change too quickly to be filtering options for route creation. These characteristics should be modeled as constraints applied to routes already developed. Naylor (2009) selected source and destination distance of the en route airfield, parking capacity, fuel capacity, diplomatic relations with the host country, proximity to seaports, number of potential destination airfields, and cargo throughput as airfield characteristics of value for en route airfield selection.

Parking capacity, fuel capacity and proximity to seaports are only constraining under certain usage scenarios and are therefore not used for en route airfield filtering. Since cargo throughput is dependent on sortie distance, pavement strength, runway length, and departure obstacles, these airfield characteristics are further analyzed as filtering opportunities. In addition to the airfield characteristics that impact cargo throughput, runway width and diplomatic relations with the host country are also analyzed as nodal reduction heuristics.

Effective runway length

A potential target for filtering airfields should be runway length. Runway length is an important airfield attribute for both takeoff and landing. The measure most often used to determine the minimum runway length required for takeoff is the critical field length (CFL). Several factors impact the CFL. These factors include gross weight, temperature, pressure altitude, thrust setting, flap setting, winds, runway slope, runway condition reading (RCR), and runway surface condition (RSC). Assuming average gross takeoff weight for a specific aircraft type based off of November 2010 data, standard day temperature, sea level pressure altitude, standard thrust setting, standard takeoff flap setting, zero winds, zero runway slope, dry RCR, and zero RSC, the required CFL for takeoff for the C-5, C-17 and C-130J is 8,500 feet, 6,500 feet and 4,500 feet respectively. These CFLs worsen as temperature and pressure altitude increase. The measure most often used as the minimum runway length required for landing is the landing distance over a 50 foot obstacle. The distance to land over a 50 foot obstacle assuming similar assumptions is 5,000 feet, 4,500 feet and 3,500 feet for the C-5, C-17 and C-130J respectively. Takeoff therefore is often more constraining than landing.

Given that takeoff CFL is often the constraining factor for required airfield length, it is the ratio of actual runway length to the takeoff CFL that is of primary concern. There are two airfield specific attributes that can aid in the calculation of takeoff CFL. These include pressure altitude and temperature. Pressure altitude is determined using airfield elevation and temperature is determined using airfield latitude and elevation from the DAFIF database. Figure 29 illustrates the impact of latitude on average monthly temperature. The average monthly temperature data for Figure 29 comes from the University of Delaware (2012) and covers the period from 1981-2010. Using the temperature data, a regression is calculated to determine

average temperature given latitude for each month. Equation 40 is the regression equation for temperature φ in degrees Celsius. Table 14 provides the regression coefficients and the adjusted R^2 for each respective month and the yearly average.

$$\varphi = \beta_0 + \beta_1\sigma + \beta_2\sigma^2 + \frac{\beta_3\sigma^3}{1{,}000{,}000} \tag{40}$$

Where:

φ = Sea level temperature in degrees Celsius
σ = Latitude in decimal degrees

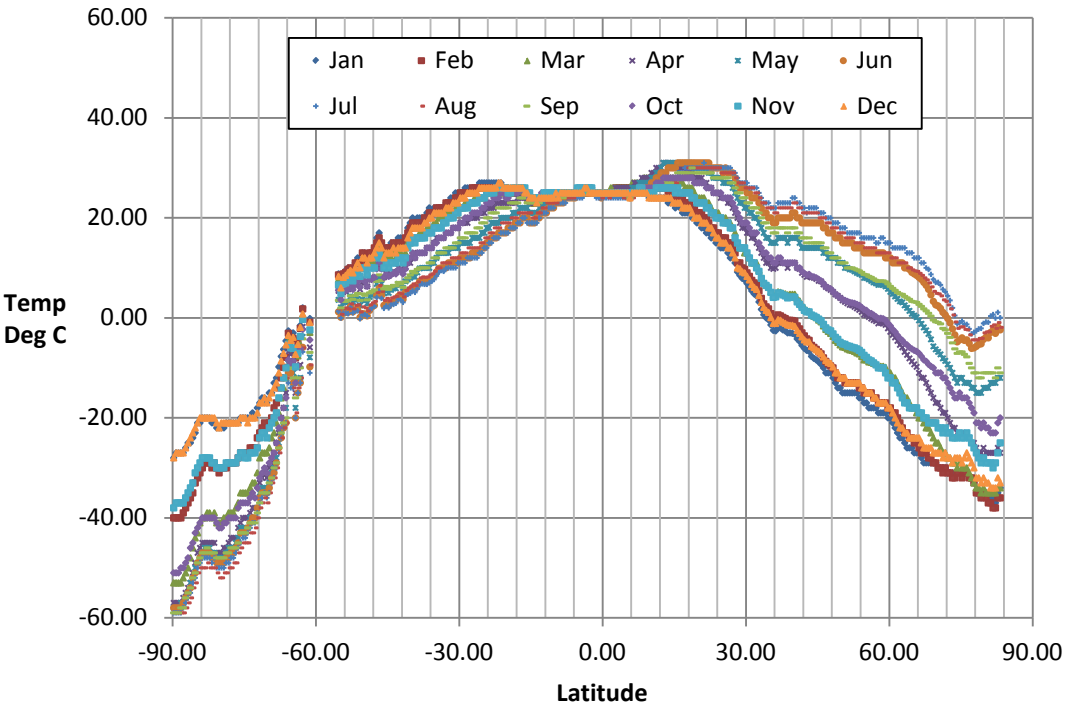


Figure 29: Average monthly sea level temperature vs latitude (1981-2010)

Table 14: Average monthly temperature $\boldsymbol{\varphi}$ regression coefficients and adjusted R^2

	β_0	β_1	β_2	β_3	Adj R^2
Jan	23.16	-0.2832	-0.0084	26.06	0.9667
Feb	24.78	-0.2575	-0.0092	30.32	0.9765
Mar	26.80	-0.1651	-0.0099	28.21	0.9824
Apr	27.67	-0.0311	-0.0097	20.00	0.9797
May	27.06	0.0825	-0.0088	15.77	0.9784
Jun	26.33	0.1696	-0.0080	12.45	0.9796
Jul	26.17	0.1984	-0.0077	12.91	0.9805
Aug	26.80	0.1565	-0.0080	18.38	0.9799
Sep	27.26	0.0703	-0.0085	22.54	0.9811
Oct	26.89	-0.0315	-0.0089	21.87	0.9805
Nov	24.95	-0.1474	-0.0086	20.51	0.9738
Dec	23.21	-0.2418	-0.0082	22.29	0.9643
Year	25.92	-0.0400	-0.0087	20.94	0.9814

Once sea level temperature has been determined, an adjustment needs to be made to correct for airfield elevation according to Equation 41. This correction is based off of the standard day model. Given this adjusted temperature φ_{adj} and the airfield elevation, the CFL for a given aircraft can be determined. Using data from Air Force Technical Order 1MDS-1-1, a regression was performed that determined CFL θ given temperature and elevation as seen in Equation 42. The coefficients and adjusted R^2 for different aircraft types are shown in Table 15.

$$\varphi_{adj} = \varphi + \beta_{Alt}\alpha \tag{41}$$

Where:

φ_{adj} = Airfield temperature in degrees Celsius
φ = Sea level temperature in degrees Celsius
β_{Alt} = -1.9812 degrees Celsius per thousand feet
α = Elevation in thousands of feet

$$\theta = \beta_0 + \beta_1 \varphi_{adj} + \beta_2 \alpha + \beta_3 \alpha^2 \quad (42)$$

Where:

θ = Critical field length in feet
α = Elevation in thousands of feet

Table 15: Critical field length $\boldsymbol{\theta}$ regression coefficients and adjusted R^2

	β_0	β_1	β_2	β_3	Adj R^2
C-5	5443	12.96	17.10	113.6	0.9787
C-17	2962	14.95	92.40	12.49	0.9981

Airfields can be filtered using the actual runway length. This is useful to remove airfields that are less than a regulation defined minimum runway length. Air Force Instruction 11-2MDS-V3 (2011) declares that the minimum runway length is 6,000 feet for the C-5, 3,500 feet for the C-17 and 3,000 feet for the C-130. Filtering airfield nodes based off these aircraft minimums results in the airfield reduction potential shown in Figure 30. Filtering on minimum runway length reduces the number of C-5 airfields by 40 percent, the number of C-17 airfields by six percent, and the number of C-130 airfields by one percent. Yet, this reduction retains many airfields that could be of low value due to a combination of high temperatures and high pressure altitudes and eliminates potentially valuable airfields at sea level and cold temperatures. Adjusting actual runway length to take into account aircraft capability with respect to both pressure altitude and temperature could provide for a superior metric to filter airfields.

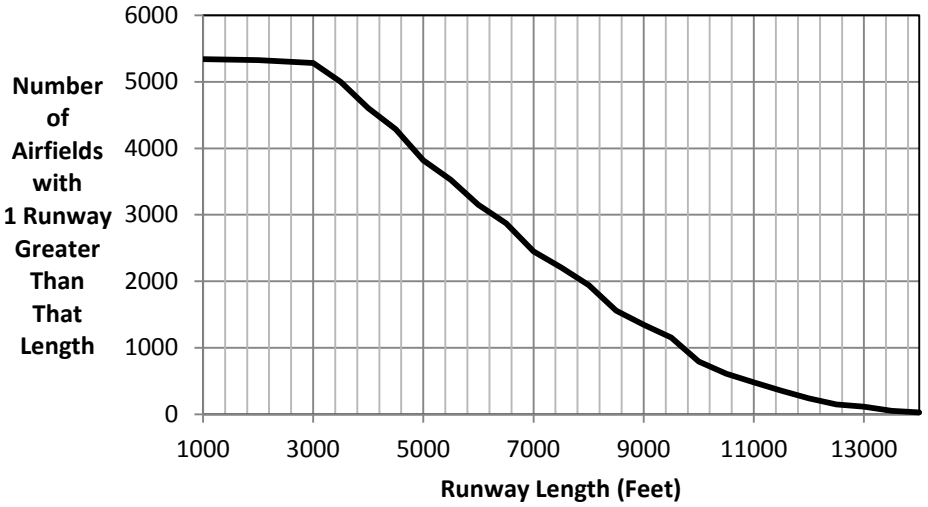


Figure 30: Number of airfields remaining vs actual runway length

Juan Mendoza (SLOR) airfield in Bolivia has an average annual temperature of 18 degrees Celsius and is located at 12,146 feet elevation. The runway at the airfield is 6,125 feet long. The CFL of a 420,000 pound C-17 at SLOR is 6,195 feet. Since the CFL exceeds the actual runway length, the aircraft is not allowed to takeoff from the airfield without reducing fuel or cargo. Compare this to Alert (CYLT) airfield in Canada which has an average annual temperature of -20 degrees Celsius and is located at 100 feet elevation. The runway at CYLT is 5,500 feet, 625 feet shorter than SLOR. Yet, the CFL of a 420,000 pound C-17 at CYLT is 2,670 feet, which is less than half that of SLOR. The CYLT aircraft could carry an additional 180,000 pounds of cargo and fuel before the CFL exceeds the actual airfield length.

To better filter airfields so that runways reflect capability, an effective runway length will be calculated from actual runway length, airfield elevation and average annual temperature. Equation 43 calculates effective runway length $\sigma_{Effective}$ by multiplying the actual runway length by the CFL at standard day temperature and sea level divided by the CFL at the airfield

latitude and elevation determined temperature and actual airfield elevation. The impact of effective runway length on the airfields in the database for the C-17 is shown in Figure 31.

$$\sigma_{Effective} = \sigma_{Actual} * \frac{\theta_{Reference}}{\theta_{Actual}} \tag{43}$$

Where:

$\sigma_{Effective}$ = Effective runway length
σ_{Actual} = Actual runway length
$\theta_{Reference}$ = Critical field length at sea level and standard day temperature
θ_{Actual} = Critical field length from Equation 42

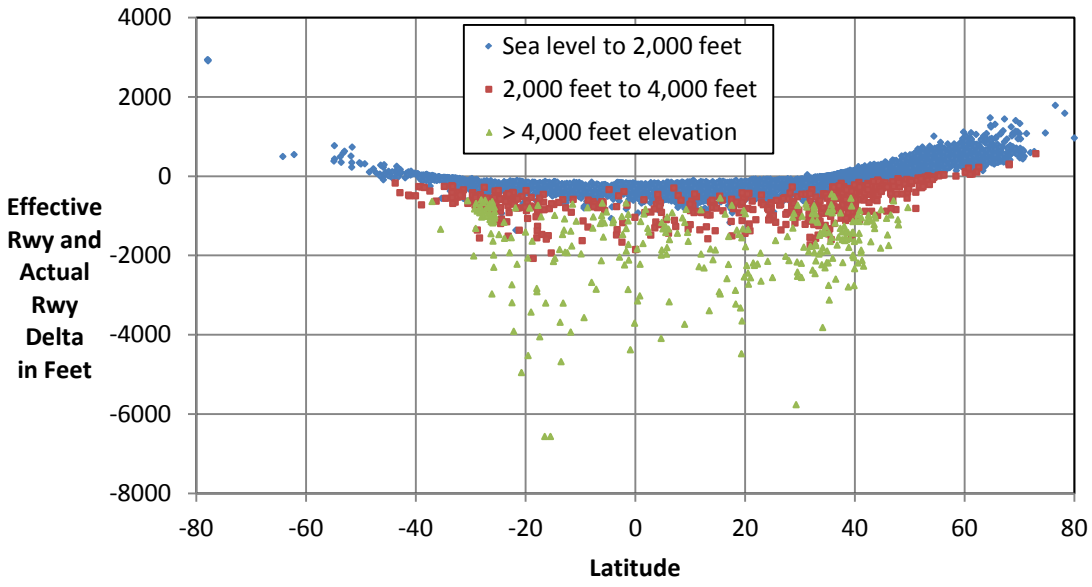


Figure 31: Impact of latitude and elevation on $\boldsymbol{\sigma_{Effective} - \sigma_{Actual}}$

Returning to the examples of SLOR and CYLT, the actual runway length for SLOR is 6,125 feet and for CYLT is 5,500 feet. The effective runway length for these airfields using Equation 43 is 3,296 feet for SLOR and 6,739 feet for CYLT. If we filter based off the minimum runway length for the C-17 of 3,500 feet using actual runway length, both airfields

would have been retained, but by using effective runway length instead of actual, the low value airfield SLOR would have been removed. Although the percentage of airfields removed for the effective runway is similar to the percentage removed using the actual runway, the effective runway length metric ensures that the appropriate low valued airfields are filtered out.

Runway width

Not only are specific aircraft limited by regulation to a minimum runway length, they are also limited to a minimum runway width. According to Air Force Instruction 11-2MDS V3, the C-5, C-17 and C-130 have minimum runway widths of 147 feet, 90 feet, and 80 feet respectively. Removing airfields below these minimum requirements would ensure that low value options are not analyzed. Figure 32 illustrates that filtering on runway width alone reduces the number of airfields by 47 percent for the C-5, 14 percent for the C-17, and 13 percent for the C-130. When filtering on the length and the width simultaneously, many of the airfields removed are the same. Filtering on runway width differs from runway length in that there is a takeoff weight benefit to additional runway length that does not exist with additional runway width. This suggests that the aircraft specific minimum runway width from regulation would be the primary and likely only criteria upon which the decision maker would filter.

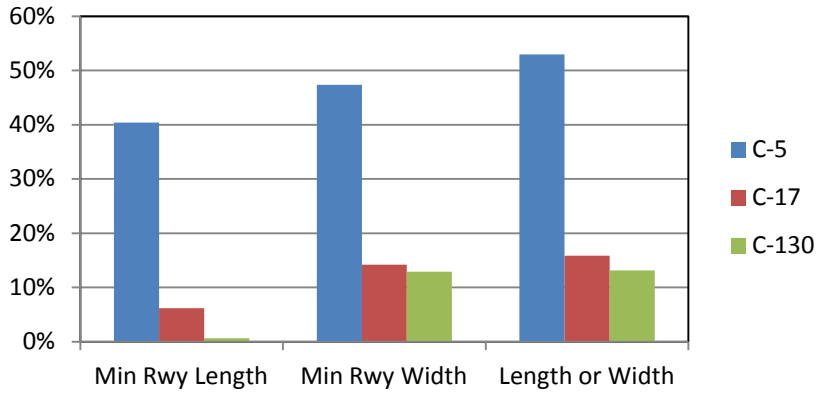


Figure 32: Airfield percent reduction by filtering on regulation minimums

Pavement strength

Gendreau and Soriano (1998) evaluate the structural capacity of airfield pavements with the aircraft and pavement classification numbers discussed by Lurdes et al. (1990) and Alexander and Hall (1991). To filter off of pavement strength, the pavement classification number (PCN) field from the DAFIF database is used. The PCN is a seven character alphanumeric designator. The first three characters of the PCN are the PCN number. The fourth character is the type of pavement and can either be R for rigid or F for flexible. The fifth character is the pavement subgrade category and includes high (A), medium (B), low (C) and ultra-low (D). The sixth character represents the highest tire pressure authorized and the seventh character designates whether the classification came from aircraft experience or technical evaluation. Using the first five characters, the maximum weight of the aircraft for that pavement can be determined.

The relationship between the PCN number and maximum aircraft weight is linear. Equation 44 shows that relationship. For each aircraft, pavement and subgrade type there is an associated slope and intercept as shown in Table 16. These values were determined using the aircraft classification number (ACN) from Air Force Pamphlet 10-1403. Martin and Voltes-Dorta (2011) use maximum gross takeoff weight as a proxy for ACN. Airfields can then be filtered based on the selected weight for a given aircraft type. Of the 5,342 airfields in the DAFIF database, 3,000 airfields had PCN values for the longest runway other than NULL. We illustrate the impact on airfield reduction of using operating weight and the weight required to fly 50 NMs (an example short distance sortie) for the three aircraft types in Figure 33.

$$\omega = \omega_{acn} * \rho + \epsilon_{acn} \quad (44)$$

Where:

ω = Aircraft maximum gross weight on pavement
ω_{acn} = Aircraft specific change in weight per change in ρ
ρ = PCN number (first three characters)
ϵ_{acn} = Aircraft specific maximum weight at zero ρ

Table 16: Aircraft maximum weight parameters

	C-5		C-17		C-130	
Pavement/Subgrade	ω_{acn}	ϵ_{acn}	ω_{acn}	ϵ_{acn}	ω_{acn}	ϵ_{acn}
FA	17.26	201.41	8.91	121.59	3.75	62.50
FB	15.53	172.07	7.77	126.62	3.46	57.31
FC	12.59	159.89	6.18	145.96	3.46	46.92
FD	8.32	174.29	4.59	153.45	3.10	41.55
RA	22.19	196.48	10.10	59.80	3.46	57.31
RB	35.85	15.54	10.10	59.80	3.21	56.07
RC	16.64	190.93	10.10	59.80	3.00	52.00
RD	13.71	182.12	6.59	123.91	2.90	50.16

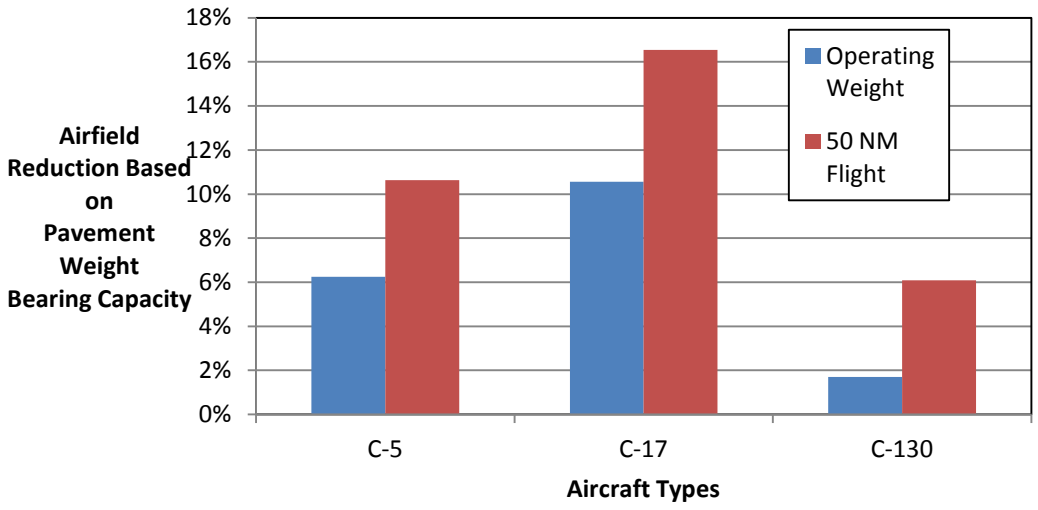


Figure 33: Airfield reduction based on pavement strength and aircraft gross weight

Analyzing the overlap between runway length, width and pavement filtering results in 98 percent of C-5 airfields, 20 percent of C-17 airfields, and zero percent of C-130 airfields removed using the pavement filter having been previously removed with the runway length or width filters.

Departure obstacles

Given that the runway is of sufficient length and width and the pavement is of sufficient strength for an aircraft, the maximum weight for takeoff of that aircraft can still be negatively impacted by departure obstacles. Certain runways require specific climb gradients to ensure adequate obstacle avoidance. These climb gradients are not entered as a field inside the runway table of DAFIF, but instead are included in a comment block on a separate table. Of the 5,342 airfields in the database, 1,294 have an obstacle that protrudes through the 40 to one obstacle clearance surface (OCS). The OCS is a cone established at the end of the runway through which no obstacles should protrude. If an obstacle protrudes this surface, climb gradient information required to clear the obstacle is published. Of the 1,294 airfields, 313 require a climb gradient higher than the standard of 200 feet per NM to ensure obstacle avoidance. These climb gradients range from 201 to 776 feet per NM. In addition to decreasing the maximum gross takeoff weight, higher climb gradients increase the risk of pilot error resulting in controlled flight into terrain (CFIT) accidents. Oster, Strong and Zorn (2013) show that pilot error including CFIT is responsible for 20 percent of aviation fatalities from 1990 to 2011.

To better illustrate the impact of climb gradients, Jackson Hole airfield in Wyoming is used as an example. Jackson Hole is the site of a 1996 CFIT accident. Jackson Hole airfield has a large mountain to the north of the airfield at 13,770 feet. This mountain penetrates the OCS and therefore requires a climb gradient in excess of 200 feet per NM to clear. The actual climb

gradient published to clear this obstacle is 450 feet per NM up to 14,000 feet. For an aircraft to meet this climb gradient at the airfield's elevation and temperature, the gross weight at takeoff must be limited. An aircraft can land at this airfield with a heavier gross weight, but would be unable to leave due to the climb gradient. This would make the airfield a low value option as an en route selection.

In order to assess the value limitation imposed by departure obstacles, actual aircraft climb gradient capability is calculated. The climb gradient for the C-17 was determined using the regression formula in Equation 45. The regression was based off of data from the climb-out factor charts in the Technical Order 1-MDS-1-1. Table 17 shows the parameters and adjusted R^2 of this regression. Using the maximum climb gradients for each of the 313 airfields, the maximum gross weight at takeoff for the C-17 is calculated as shown in Equation 46. The C-17 maximum gross takeoff weight is plotted against the climb gradient in Figure 34.

$$\tau = \beta_0 + \beta_1 \varphi_{adj} + \beta_2 \alpha + \beta_3 \omega \tag{45}$$

Where:

τ = Climb gradient in feet per NM
φ_{adj} = Temperature in degrees Celsius
α = Elevation in thousands of feet
ω = Maximum aircraft gross takeoff weight in thousands of pounds

Table 17: Climb gradient $\boldsymbol{\tau}$ regression coefficients and adjusted R^2

	β_0	β_1	β_2	β_3	Adj R^2
C-17	1991	0.5571	-28.90	-3.262	0.9853

$$GW = \frac{\tau - \beta_0 - \beta_1 \varphi_{adj} - \beta_2 \alpha}{\beta_3} \tag{46}$$

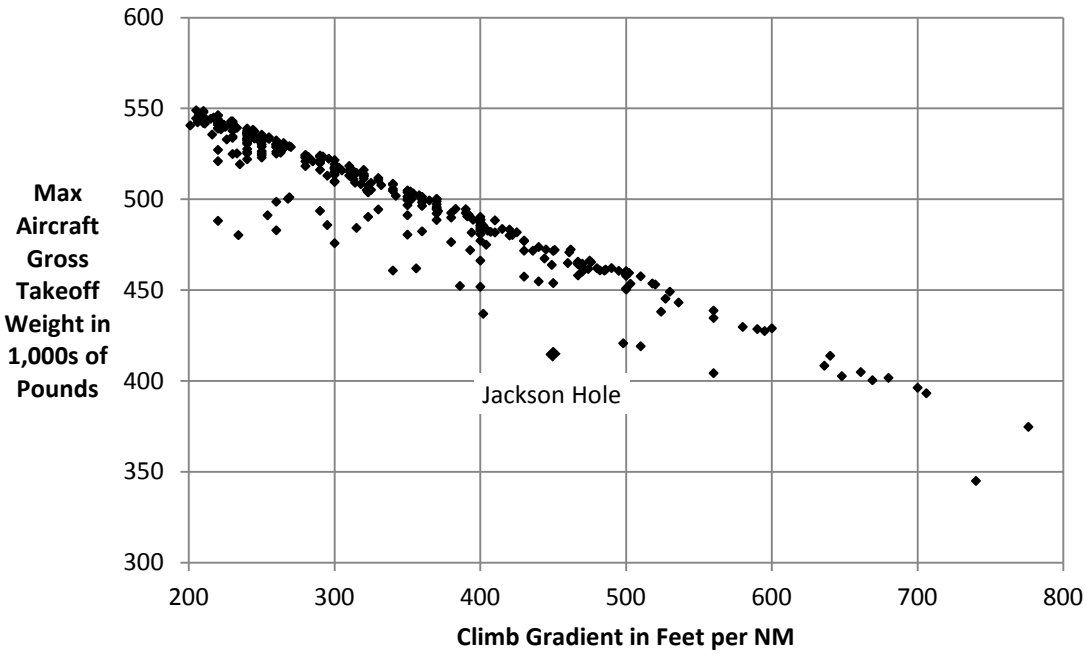


Figure 34: C-17 maximum gross takeoff weight vs climb gradient $\boldsymbol{\tau}$

Although airfields can be filtered based on departure obstacles using the maximum gross takeoff weight available at a given airfield, the level of nodal reduction is relatively modest. For example, if we were to remove all airfields unable of achieving the average maximum gross takeoff weight for the C-17 of 440,000 pounds, then only 22 airfields or 0.41 percent would be removed. In addition, other runways might be available at the airfield, which have a lower climb gradient then the maximum for the airfield. Using the lowest climb gradient runway would result in an even smaller nodal reduction. Despite the inherent weakness in using climb gradient information for filtering, it does provide critical information for route value determination.

Diplomatic clearances

The final airfield characteristic heuristic for nodal reduction is the diplomatic clearance heuristic. Sere (2005) listed diplomatic relations with the host country as one of his six major factors when determining the locations of en route airfields. Some airfields are located in

countries where political considerations make it difficult to obtain a diplomatic clearance for aircraft to enter that country. If a decision maker decides that certain countries are not politically viable, then the airfield nodes in those countries can be removed from consideration in route generation.

There are airfields in several countries that are not optimal for selection as en routes. The reasons these countries lack value often arises from the political sensitivities involved. Figure 35 shows the percentage impact of removing these countries from the set of available en route airfields. The total reduction from removing these airfields is 5.6 percent. The technique of removing airfields based on political sensitivities can increase the speed of computation of routing alternatives, but eliminates potential high value routes. Use of the technique should be limited to only those countries where the use of en route airfields is politically untenable.

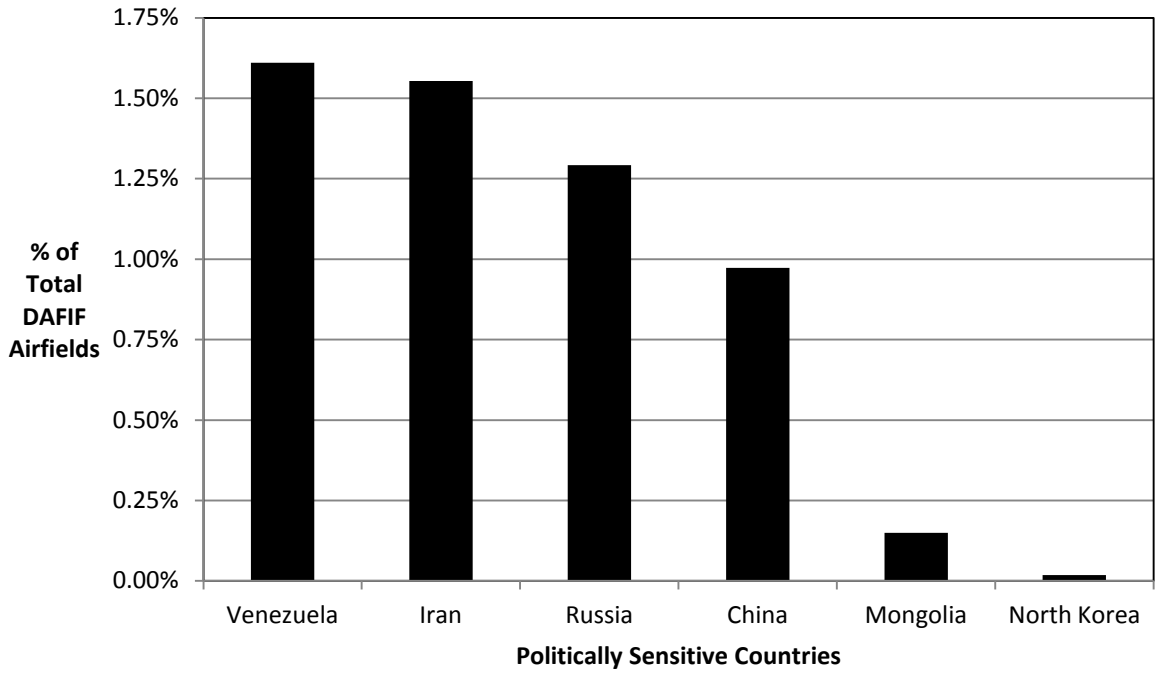


Figure 35: Politically sensitive countries as percentage of total

Combined nodal reduction

Combining all nodal reduction heuristics results in significant overall nodal reduction. For example, starting with 5,342 airfields and filtering out airfields whose longest runway is less than the C-17 runway length minimum of 3,500 feet results in 5,010 airfields. After filtering on runway length, we filtered on the C-17 minimum runway width of 90 feet which returned only 4,549 airfields. Following the runway width filter, all airfields with a maximum effective runway length less than 5,000 feet were removed, ending with 3,725 airfields. The next filter applied is the pavement strength filter removing all airfields which cannot support a 500,000 pound gross weight takeoff or less, bringing the total number of airfields remaining to 2,581. The final airfield characteristic heuristic applied is the country heuristic. Removing airfields from Russia, China, Iran, Venezuela, Mongolia and North Korea leaves 2,427 airfields. Use of these combined heuristics culminates in an over 50% nodal reduction.

The difference between applying the cutoff distance model to the set of 2,427 airfields contrasted against the original 5,342 airfields for each of the 100 OD pairs from Table 13 can be seen in Figure 36. Note the significant airfield reduction at higher distances. This nodal reduction is achieved without the use of the total distance multiple. If a decision maker wants to select among a given number of alternatives, a given n and max k can provide that desired number. From the potential en route airfields for a given requirement, only the top ranked by total distance multiple can be selected as primary en route airfields to achieve the desired overall number of alternatives. For each primary en route airfield, it is also possible to select the top number of secondary en route airfields using total distance multiple to meet the targeted number of alternatives.

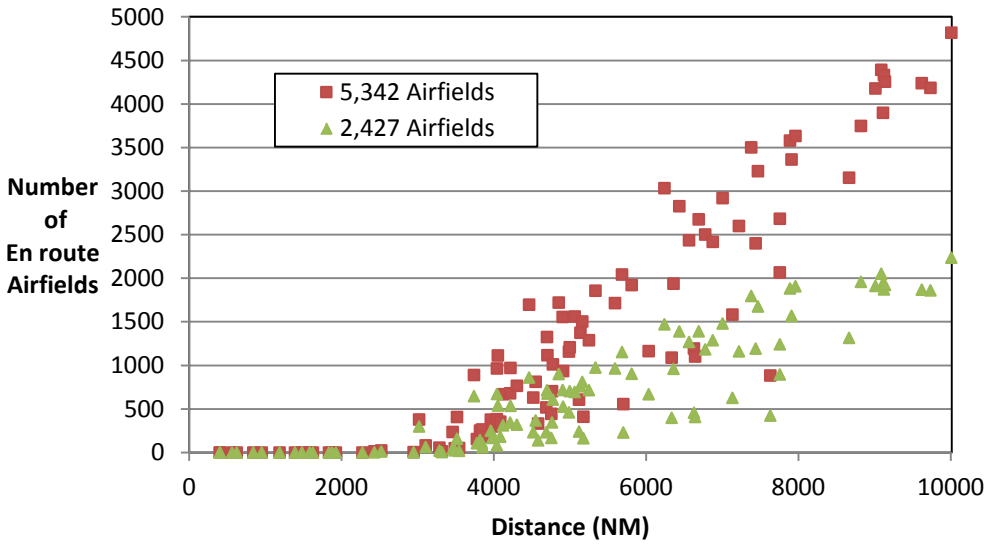


Figure 36: Impact of combined heuristics on en route airfield reduction

Speed and Accuracy

There is a tradeoff using nodal reduction between speed and accuracy. As nodes are removed from consideration, speed is increased but accuracy has the potential to decline. To understand the extent of this tradeoff for the developed heuristics, 27 OD pairs were selected at regular spaced distance intervals. The assumptions for the "with nodal reduction" technique include the selection of the C-17 as the MDS, a 7,000 feet minimum effective runway length, a minimum pavement strength of 460,000 pounds, a maximum number of 200 primary airfields and a maximum number of 20 secondary airfields. This was contrasted against using all nodes within the initial "eye shape" formed between source and destination. Due to RAM limitations of the workstation for the "without nodal reduction" technique, the number of legs for analysis was limited to three. The "with nodal reduction" technique was never limited by RAM for any OD pair tested. The maximum number of legs coded for in the algorithm was five.

The nodal reduction techniques for the sample resulted in 99.44% route planning cargo throughput accuracy using only 4.26% of the computation time. 100% accuracy could have been

achieved by reducing the minimum effective runway length to 5,000 feet and increasing the maximum number of secondary airfields to 35. Of the 27 OD pairs computed for using both the "with nodal reduction" and "without nodal reduction" techniques, 22 selected the exact same route for optimal cargo throughput. Four of the "with nodal reduction" routes with lower cargo throughput were due to the removal of airfields with effective runways lower than 7,000 feet. The final "with nodal reduction" route with lower cargo throughput was lower due to the limit on the number of secondary airfields to 20.

The impact of nodal reduction on speed can be seen in Figure 37. Using nodal reduction, the longest time was under 7 seconds and without using nodal reduction, the longest time was over 4 minutes. The run that resulted in over 4 minutes computed 680,191 routes for comparison. RAM limitations prevented analysis beyond this point. Reduction in computation time increased with OD pair distance. The longest distance run for "with nodal reduction" took only 2.4% of the computation time of "without nodal reduction." The average time reduction was over 96% for the 27 OD pairs.

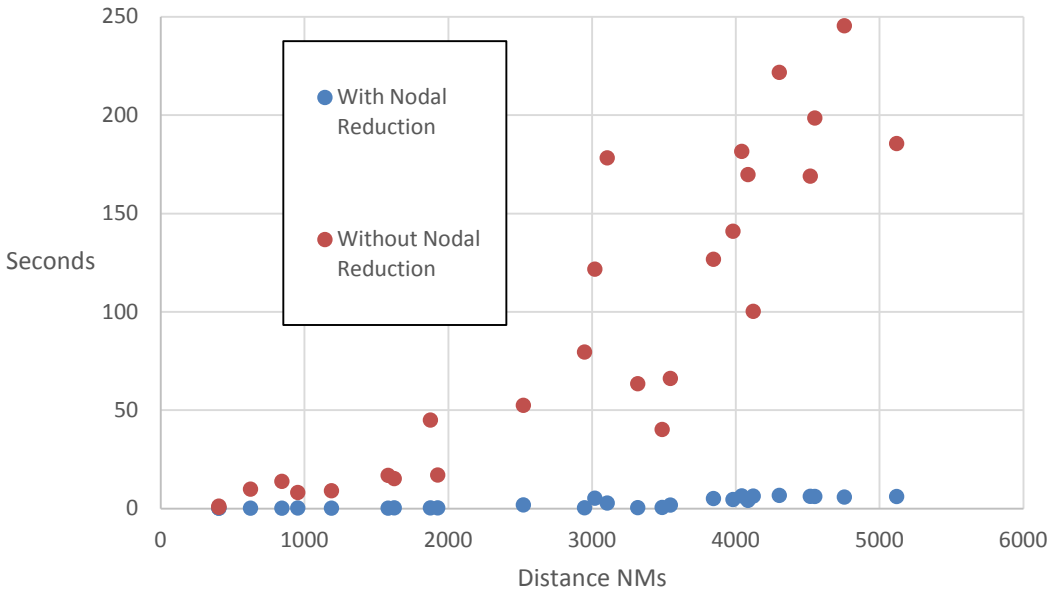


Figure 37: Time Comparison for Route Analysis

The impact of nodal reduction on accuracy can be seen in Figure 38. The level of accuracy for the analysis achieved cargo throughputs that were on average 99.4% of the optimal level set by the "without nodal reduction" case.

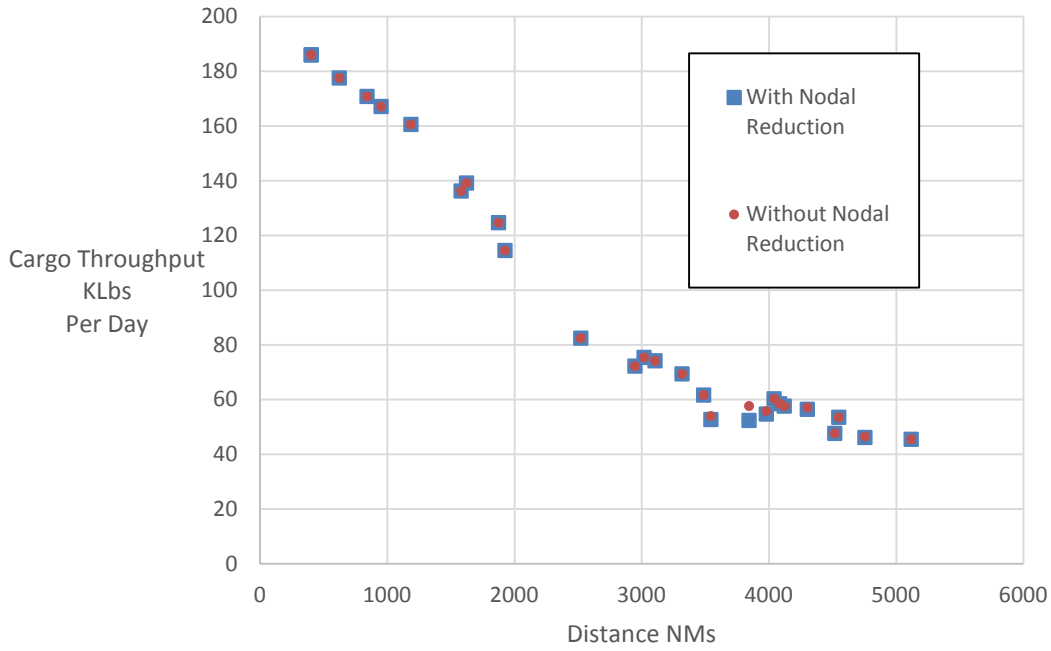


Figure 38: Cargo Throughput of Optimal Route Analysis

Conclusion

Nodal reduction can result in an over 95% decrease in computation time for less than a one percent loss in accuracy. This tradeoff between speed and accuracy can be altered as the situation dictates. Certain potential alternatives that are eliminated are of sufficiently low value that they are not worthy of consideration. In this situation, computation speed is attained without sacrificing the quality of the decision being made. Many situations require rapid routing analysis. These time sensitive situations can occur from a humanitarian crisis, a natural disaster

or a military conflict. After the initial routes are developed and scheduled, the tradeoff between speed and accuracy can shift to accuracy as time allows for more extensive computation.

Upon completion of nodal reduction and route creation, routes can be ranked on the basis of cargo throughput, fuel efficiency, time or cost. Route selection can then be easily performed to optimize enterprise airlift on any of these metrics. The airfields in the top alternatives might potentially be enhanced to improve pavement strength, runway length, fueling infrastructure, material handling equipment or maintenance capabilities. Should certain airfield nodes become unavailable due to a working Maximum On the Ground (MOG) limitation, inclement weather or airfield damage, the next optimal alternative can easily be selected. In addition, given the set of top ranked routes, the impact of aircraft type, crew complement, staging and trans-load on cargo throughput and fuel efficiency can be assessed. Nodal reduction is the critical foundation that enables route creation and assessment for the SAP.

V. Methodology

The first article established what is of airlift value and aided understanding of the metrics that could be utilized to assess that value. The second article used Value Focused Thinking (VFT) to associate value with a given flight distance to enable the creation of a distance cutoff model. The third article used a set of heuristics to reduce the set of selected airfields to only those airfields that have the best chance to create high value routes. Using the mathematical models developed from these three articles and the decision maker preferences on the tradeoff between accuracy and speed, routing alternatives can be created.

Route Alternative Generation

The algorithm used for route alternative generation and analysis was developed in Javascript and can be seen in Appendix A. The name of the function is "BuildRoutes()." The basic method involves using the set of primary airfields and each primary airfield's associated set of secondary airfields as described in the third article. Initially we increment through the desired number of stops starting at zero and proceeding to the user selected maximum. For the code created, that maximum was limited to four stops. This was chosen based on the C-17. Since optimal value distance for the C-17 is approximately 2,000 NMs and the largest randomly selected OD pair distance was 10,000 NMs, four stops would provide high valued alternatives for airfields that are near antipodal on the globe.

For every route, feasibility is addressed initially. Max range zero payload is calculated for the aircraft type selected. If any sortie on the route has a distance in excess of that max range, then that route is not created. Starting at zero stops, the requirement OD pair builds the zero stop route. For one stop routes, a loop is set up for every airfield in the primary airfield set. For each primary airfield, two sorties are built; one from the origin to the selected primary

airfield and one from the selected primary airfield to the destination. For two stop routes, two loops are established. The outer loop increments through primary airfields and the inner loop increments through the set of secondary airfields associated with that primary. For each primary and secondary airfield pair, three sorties are built; one from the origin to the primary, one from the primary to the secondary and one from the secondary to the destination.

For three stop routes, three loops are established. The outer loop increments through primary airfields, the intermediate loop increments through the set of secondary airfields associated with the primary airfield and the inner loop increments through the set of secondary airfields associated with the primary airfield that has the same ID as the secondary airfield of the intermediate loop. For each set of primary and two secondary airfields, four sorties are built; one from origin to primary, one from primary to secondary, one from secondary to tertiary and one from tertiary to destination. This process can be repeated until the number of stops established by the decision maker is reached.

After feasibility is established and the airfields that make up the route are selected. Each sortie's parameters are established using the "FuelConsumedIterationGrossWeightFixed()" prototype function of the Aircraft object as seen in Appendix B. This function is based off of the algorithm and regressions established in the second article. Sortie parameters include operating weight; fuel for start, taxi and takeoff; time, distance and fuel for climb; takeoff gross weight; payload; altitude; ramp fuel; time, distance and fuel en route; Mach and true airspeed en route; time, distance and fuel for descent; and the time and fuel for the approach. Altitude selected is optimal for the given takeoff gross weight. Sortie parameters are stored in a sortie object in Appendix A that also includes the "From" airfield id, "To" airfield id, total distance, total time, total fuel and cargo throughput. Total distance is a sum of climb, en route and descent distances.

Total time is a sum of climb, en route, descent and approach times. Total fuel is a sum of fuel for start, taxi and takeoff, climb, en route, descent and approach fuel. Cargo throughput in the sortie object is only calculated if trans-load operations are planned. The equation utilized is similar to Equation 28 and is shown in Equation 47. The actual algorithm can be seen in the Aircraft object prototype function "CargoThroughputTransload()" of Appendix B.

$$\frac{Klbs}{Day} = \frac{12 * Payload_{Max} * Sorties}{Sortie\ Cycle\ Time} \tag{47}$$

All of the sortie instances are stored in an array of sorties called srtArr[]. This sortie array is attached to a route object in Appendix A that also includes the number of stops on the route, total distance, delivery time, cycle time, maximum payload without trans-load, cargo throughput, cargo throughput for a planned payload, fuel efficiency and fuel efficiency for a planned payload. Total distance is the sum of the sortie total distances. Delivery time is based off of the Aircraft object prototype function "RouteDeliveryTimeCalculator()" in Appendix B and is dependent on crew complement, staging and trans-load decisions. Cycle time is based off of the Aircraft object prototype function "RouteCycleTimeCalculator()" in Appendix B and is dependent on crew complement, staging and trans-load decisions.

Maximum payload without trans-load is the minimum of all the sortie payloads on that route. Cargo throughput and cargo throughput for planned payload are dependent on whether trans-load operations are selected. If trans-load operations are selected, then Equation 48 is used with *i* representing the i^{th} sortie and n representing the total number of sorties on that route. If trans-load operations are not selected Equation 49 is used with max payload without trans-load and route cycle time described above.

$$Route\ Cargo\ Throughput\ Transload\ in\ \frac{Klbs}{Day} = \frac{Max(Sortie\ Cargo\ Throughput_i)}{\sum_{i=0}^{n} \frac{Max(Sortie\ Cargo\ Throughput_i)}{Sortie\ Cargo\ Throughput_i}} \quad (48)$$

$$Route\ Cargo\ Throughput\ in\ \frac{Klbs}{Day} = \frac{24 * Max\ Payload\ Without\ Transload}{Route\ Cycle\ Time} \quad (49)$$

Fuel efficiency used when comparing routes is similar to the FEI from the first article. It is the ratio of the cargo throughput per day achievable on the route to the fuel consumed per day. The amount of fuel consumed per day for trans-load operations is calculated using the Aircraft object prototype function "DailyFuelConsumptionTrans()" in Appendix B and is based off the sortie array, augmented crew and staging decision. Calculations for fuel efficiency and fuel efficiency for a planned payload are also dependent on whether trans-load operations are selected. If trans-load operations are selected, then Equation 50 is used with route cargo throughput trans-load from Equation 48 and route fuel trans-load from above. If trans-load operations are not selected, Equation 51 is used with route cargo throughput from equation 49, n representing the total number of sorties, *i* representing the ith sortie on that route and route cycle time as described above.

$$Fuel\ Efficiency\ Transload = \frac{Route\ Cargo\ Throughput\ Transload}{Route\ Fuel\ Transload} \quad (50)$$

$$Fuel\ Efficiency = \frac{Route\ Cargo\ Throughput}{\frac{24 * 2 * \sum_{i=0}^{n} Sortie\ Fuel_i}{Route\ Cycle\ Time}} \quad (51)$$

Route Comparison

Each route object instance is then stored in an array called routes[]. The routes in the array have several measures over which they can be sorted to ascertain the optimal value for that measure. These measures include distance, maximum payload, cycle time, cargo throughput for maximum payload, cargo throughput for a planned payload, fuel efficiency for maximum payload and fuel efficiency for a planned payload. Out of these measures, two are of particular importance. This was recognized in the weightings of the value model of the second article. The two measures of greatest importance are cargo throughput and fuel efficiency.

The route alternative generation requires several selections by the user that can influence the results. These selections include the aircraft type, crew complement, staging of crews and trans-load operations. The impact of these selections on cargo throughput and fuel efficiency will be examined in more detail. To simplify analysis, out of the 100 OD pairs from the third article, 27 OD pairs were selected at evenly spaced distances. In addition, several commonly travelled routes by Air Mobility Command are examined with the route analyzer for managerial implications.

VI. Results

Analysis of the routes was performed by selecting the top route for a given OD pair on the basis of cargo throughput. The cargo throughput for that top route was recorded. The top route was then selected based on fuel efficiency and that fuel efficiency was recorded. To calculate the top route, a common set of assumptions were used. These assumptions include an effective runway length greater than 5,000 feet, a minimum actual runway length and width greater than that aircraft's regulation minimum, CFL computed based on the aircraft's maximum possible gross takeoff weight, runway strength capable of supporting that aircraft's maximum possible gross takeoff weight, a maximum of 200 primary airfields, a maximum of 20 secondary airfields for each primary and a maximum of 5 sorties.

Using these base assumptions, cargo throughput and fuel efficiency were analyzed by varying aircraft type, crew complement, staging and trans-load. The aircraft types that were contrasted include the C-5B, C-17A and C-130J. For the crew complement, staging and trans-load analysis, the C-17 was the aircraft type that was selected. The crew complement was either normal or augmented. Staging could be either with or without staging. With staging would require crews to be staged at every airfield where a crew would require crew rest in the without staging case. Trans-load was either with or without trans-load. With trans-load assumes mission handling equipment and a cargo yard are available at every airfield.

Aircraft Type

The results from the aircraft type analysis for cargo throughput can be seen in Figure 39. As distance increases, the top route's cargo throughput declines. This decline appears steep initially. Yet, it becomes more linear after 4,000 NMs. The cargo throughput of the top route

assumes the aircraft can take its maximum allowable cargo weight. If the load factor is not maximized, then the cargo throughput is significantly diminished. The C-5 appears to have the greatest capacity for cargo throughput. Yet, this capacity fails to take into account the aircraft's mission capable rate. If mission capable rates of 55%, 85% and 75% for the C-5, C-17 and C-130 respectively are taken into account then the cargo throughput would be adjusted as seen in Figure 40. Note that with mission capable rates included, the C-17 provides the greatest capacity for cargo throughput.

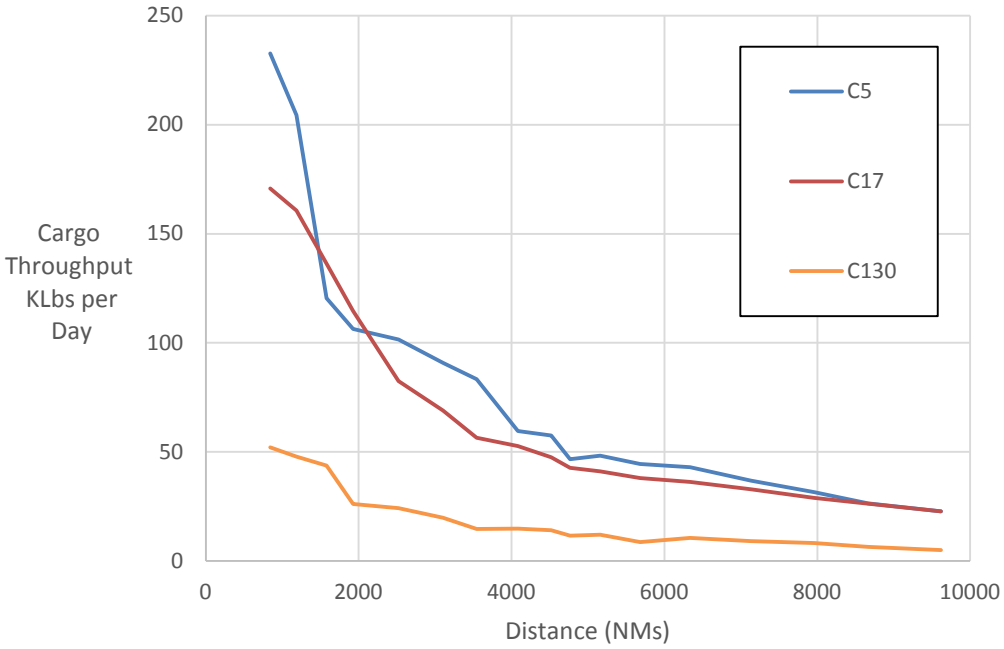


Figure 39: MDS cargo throughput and distance

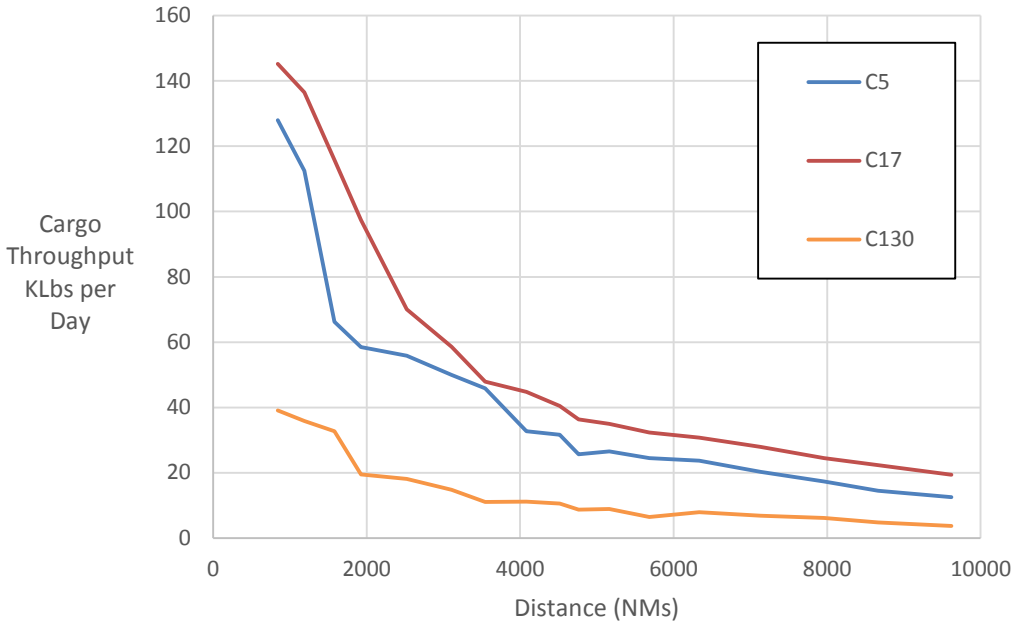


Figure 40: MC rate adjusted MDS cargo throughput and distance

The relationship between the top route's cargo throughput and distance is very similar to the relationship between the top routes fuel efficiency and distance as can be seen in Figure 41. The difference is that the three MDSs differ very significantly on cargo throughput, but are far more similar when it comes to fuel efficiency. The C-130 just barely edges out the C-5 and C-17 with the C-5 being the least fuel efficient, assuming 100% load factors. If the C-5 and C-17 are not using their full load factor, then that gives an increasing edge to the C-130 in fuel efficiency. For example, given 6 pallets weighing 5,000 pounds moving from Dover AFB (KDOV) to Ramstein AB (ETAR), the C-5, C-17 and C-130 would have a fuel efficiency of 0.07, 0.09 and 0.29 respectively. The C-130 would triple both a C-5 and C-17 in fuel efficiency given only 6 pallets weighing 30,000 pounds.

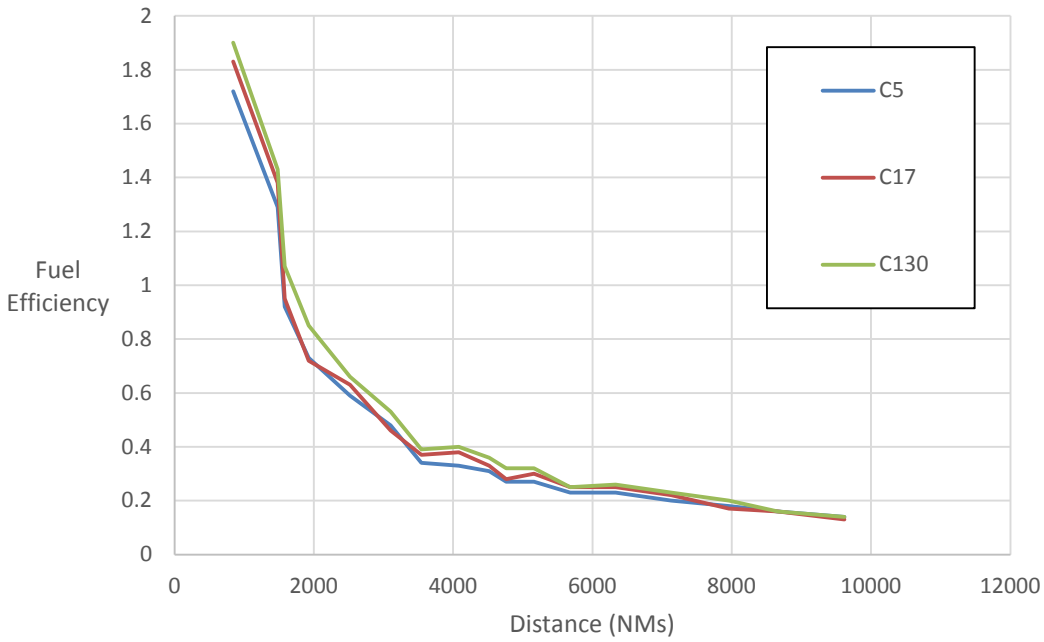


Figure 41: Fuel efficiency and distance

Crew Complement

Crew complement had a minor effect on cargo throughput as can be seen in Figure 42, and had a negligible effect on fuel efficiency. The largest increase of 36% in cargo throughput was seen on the LIBP to HCMH OD pair. The average increase is 14.4%. Augmenting crews is an option for increased cargo throughput, but if crew availability is limited, greater cargo throughput gains would likely be achieved through staging.

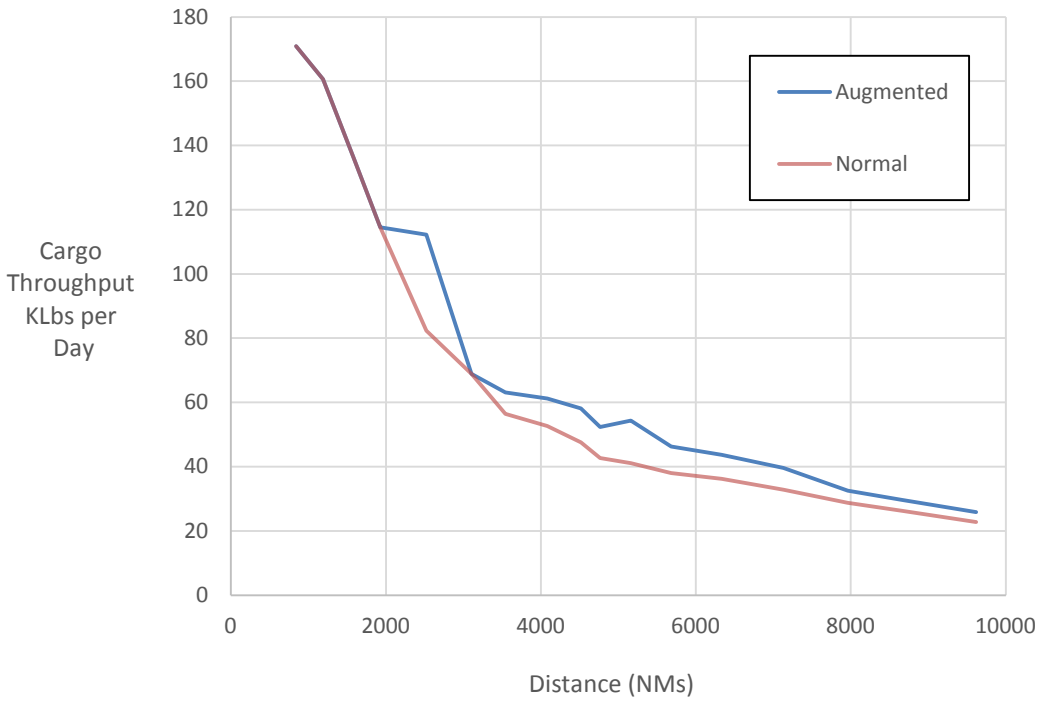


Figure 42: Crew complement cargo throughput and distance

Staging

No other factor had as great an impact on cargo throughput as staging. Simply by swapping crews when a crew runs out of crew duty day, cargo throughput capability increases on average 101.4% as seen in Figure 43. This doubling of cargo throughput decreases closure times, increases aircraft availability and enhances operational flexibility. Staging requires additional aircrew management which can add to the complexity of a mission. Staging has no impact on fuel efficiency for the increase in cargo throughput is offset by the increase in fuel consumption.

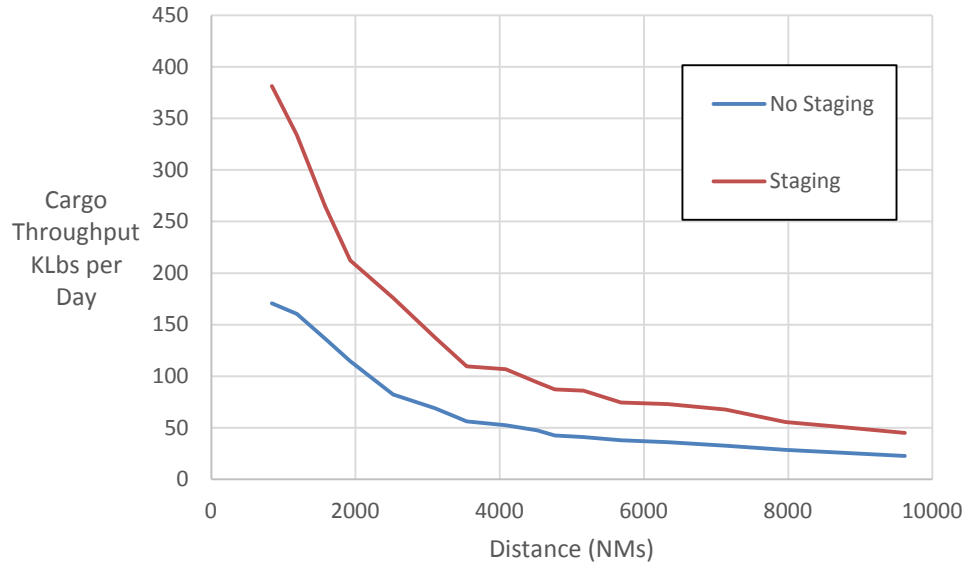


Figure 43: Staging cargo throughput and distance

Trans-load

Trans-load operations have relatively minor effects on cargo throughput as shown in Figure 44. Many times trans-load operations have a negative impact on cargo throughput due to the increased ground time associated with cargo loading and unloading. Only a third of the sample had trans-load result in an increase in fuel efficiency. When that increase occurred it was on average 8%. If the amount of fuel that goes into performing trans-load operations is included, then most trans-load operations would be less fuel efficient.

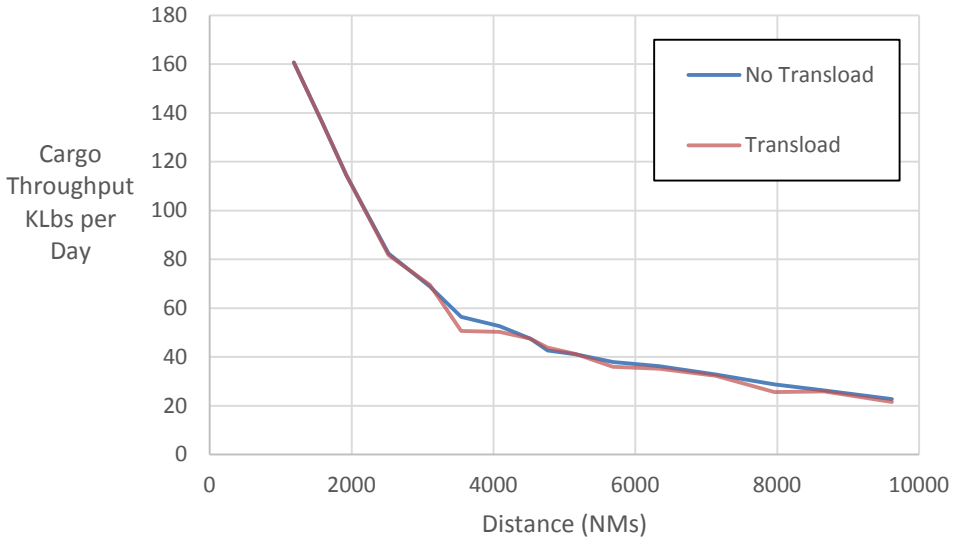


Figure 44: Trans-load cargo throughput and distance

Air Mobility Command Routes

Two routes were analyzed using the algorithms/models from the methodology and the three articles. These routes only selected airfields that are over 7,000 feet in effective runway length and that have pavement strong enough to handle the max gross takeoff weight for the aircraft. The aircraft selected for this analysis is the C-17. The first route is an East bound route coming from Dover AFB (KDOV) on the East coast and going to Bagram AB (OAIX) in Afghanistan. The second route is a West bound route coming from Travis AFB (KSUU) on the West coast and going to Osan AB (RKSO) in Korea. Since the actual requirements may not allow for the maximum payload to be loaded, the top route for lighter pallet loads are also displayed.

Table 18 contrasts the common AMC route going East of Dover (KDOV) to Ramstein (ETAR) to Bagram (OAIX) against the algorithm calculated most fuel efficient route of Dover (KDOV) to Goose Bay (CYYR) to Reykjavik (BIKF) to Lulea (ESPA) to Ufa (UWUU) to

Bagram (OAIX). These routes can also be visually compared in Figure 45 with the AMC route in orange and the fuel efficient route in red. The top fuel efficient route carries a maximum payload twice that of the regular AMC route, airlifts 73% more cargo throughput, is 73% more fuel efficient and is 45% cheaper per pound delivered. This assumes being able to load 9,494 pounds per pallet position. If only 6,000 pounds per pallet position could be loaded, then the improvement is reduced. The 6K per pallet Dover (KDOV) to Trois-Rivieres (CYRQ) to Evenes (ENEV) to Bagram (OAIX) route carries 23 thousand pounds more cargo per trip, achieves 18% more cargo throughput, is 27% more fuel efficient and is 21% cheaper per pound of cargo delivered.

Table 18: Top route comparison going East

East Coast To Afghanistan							
Route		Max Payload	Max Pounds Per Pallet	Cargo Thru	Cycle Time	Fuel Efficiency	Cost Per Pound
AMC Route	KDOV ETAR OAIX	85.38	4,743	22	95	0.15	$3.06
Top Fuel Efficiency	KDOV CYYR BIKF ESPA UWUU OAIX	170.9	9,494	38	109	0.26	$1.69
% Change from Current		**100%**	**100%**	**73%**	**15%**	**73%**	**-45%**
8K Pallet Top Fuel Eff	KDOV KPQI BIKF UUEE OAIX	150.74	8,374	35	104	0.24	$1.84
6K Pallet Top Fuel Eff	KDOV CYRQ ENEV OAIX	108.31	6,017	26	99	0.19	$2.41

Figure 45: East route (orange) and top fuel efficient route (red)

Table 19 contrasts the AMC route going West of Travis (KSUU) to Hickam (PHIK) to Guam (PGUA) to Osan (RKSO) against the most fuel efficient route from the analysis of Travis (KSUU) to Terrace (CYXT) to Shemya (PASY) to Osan (RKSO). The top fuel efficient route as shown in Figure 46 carries 50% more cargo per trip, delivers 89% more cargo throughput, is 100% more fuel efficient and is 49% cheaper per pound delivered. This assumes being able to load 7,556 pounds per pallet position. If only 6,000 pounds per pallet position could be loaded, then the improvement is reduced. The 6K per pallet Travis (KSUU) to Shemya (PASY) to Osan (RKSO) route carries 22 thousand pounds more cargo per trip, achieves 67% more cargo throughput, is 85% more fuel efficient and is 45% cheaper per pound of cargo delivered. Both east and west coast routes can significantly increase their cargo throughput per aircraft by establishing staging locations for aircrews along the routes.

Table 19: Top route comparison going West

West Coast To Korea							
Route		Max Payload	Max Pounds Per Pallet	Cargo Throughput	Cycle Time	Fuel Efficiency	Cost Per Pound
AMC Route	KSUU PHIK PGUA RKSO	90	5,000	18	119	0.13	$3.43
Top Fuel Efficiency	KSUU CYXT PASY RKSO	136	7,556	34	95	0.26	$1.74
% Change from Current		**51%**	**51%**	**89%**	**-20%**	**100%**	**-49%**
6K Pallet Top Fuel Eff	KSUU PASY RKSO	112	6,222	30	89	0.24	$1.90

Figure 46: West route (orange) and top fuel efficient route (red)

VII. Conclusions

Given a set of cargo requirements and a set of aircraft for the SAP, a critical first step is to determine the aggregation of those requirements for each aircraft. Ascertaining cargo weight and volume limitations must precede cargo aggregation. Yet, cargo weight limits are dependent on the route from the requirement origin to destination. This route selection is therefore an essential step before aggregating cargo. Since route analysis is a fundamental foundation for calculation of the SAP, establishing what is of value when comparing routes becomes a necessity. "Competitive advantage and aviation fuel efficiency" examined several metrics in an operational setting for the evaluation of airlift and assessed their inter-relationships. This paper highlighted the importance of fuel efficiency as a measure for airlift value.

"Distance value model for nodal reduction of the strategic airlift problem" developed a method for associating value with distance. The concept suggested that each aircraft type has unique capabilities that make the flight of certain distance regimes more valuable than others. If certain distance regimes are of significantly low value, perhaps airfields within those distance regimes do not have to be included when comparing routes. This value model was later validated by comparison of the top routes by cargo throughput and fuel efficiency. Despite the success of the value model, route generation was too sluggish and the number of potential route combinations became untenable. This highlighted the need for further heuristic techniques to increase computation speed without the loss of valuable alternatives.

The solution to the speed issue was resolved with "Nodal reduction heuristics applied to route generation for enterprise airlift evaluation." This offered several techniques to remove airfields from the set of potential airfields for route generation. These techniques worked well while simultaneously achieving other airlift value not established in the distance value model.

Computation times were reduced over 95% and all OD pairs that previously would fail due to reaching RAM limitations, could now be analyzed. Nodal reduction made route generation possible. Given the set of possible routes, the primary measure of effectiveness, cargo throughput, and the primary measure of efficiency, fuel efficiency, for each route was calculated. This enabled the optimal route for each measure to be selected.

The impact of distance, aircraft type, crew complement, staging and trans-load on cargo throughput and fuel efficiency highlighted the tradeoffs involved. The concept that increasing distance reduces cargo throughput is not surprising, but given a specific distance and aircraft type, an airlift planner could very rapidly assess the upper limit for cargo throughput and the lower limit for closure time. Of the three MDS aircraft types analyzed, the C-17 is the obvious mode of choice for maximizing cargo throughput. When assessing the tradeoff between using augmented crews or setting up staging operations, the choice to set up staging operations for increased cargo throughput is apparent. Finally, trans-load operations often failed to outperform no trans-load operations. This surprising finding was primarily due to increased ground times.

Applying the algorithm to some real world operational examples, opportunities to enhance cargo throughput and fuel efficiency or reduce cost were abundant. In both the East and West cases the current operational tendency was to plan routes at significantly lower latitudes than the great circle path between origin and destination. This is potentially the result of the distortion associated with a two dimensional projection of the globe. The optimal number of sorties on a route is associated with the distance involved and the payload. For the C-17, flying direct from Dover (KDOV) to Ramstein (ETAR) is only effective and efficient if the payload is limited to 94 thousand pounds or 5,222 pounds per pallet position. If it is possible to load more than that, then it is more effective to fly KDOV to Gander (CYQX) to ETAR.

The route algorithm calculates the maximum payload possible. If the aircraft can load cargo to that amount, then that is the most effective and efficient operationally. If the cargo is unable to be aggregated to that amount due to sub-volume constraints or "available to load" constraints, then either the flight should be delayed to make aggregation possible or the route algorithm should be run again with the planned load to determine the new optimal route. Delaying to make aggregation possible is a very difficult decision. It has the potential to vastly improve efficiency but comes at the risk of effectiveness.

The abstractions provided by this research help identify opportunities for improvements in efficiency and effectiveness. Several factors that affect the value of given routing alternatives that are not captured in this analysis include the actual flight routing and altitude profile on a sortie, the actual flight winds/weather experienced on a sortie and tail specific fuel consumption for a sortie. Each of these factors represent an area for future research to more accurately assign value to a given route alternative.

With the optimal route payload provided, the stage is set for the development of both a cargo aggregation algorithm and an aircraft selection algorithm. Cargo aggregation is both an aerial port "pallet" consideration and a hub and spoke consideration. It involves cargo currently in information systems and anticipated cargo based on historical trends. It is tightly tied to customer needs by their "earliest arrival date" and "required delivery date." Many facets of the cargo aggregation problem are tied to aircraft type and specific tail selection, including pallet positions available and the goal of inactive mile or "flying empty" minimization. These cargo aggregation and aircraft selection algorithms can then feedback new OD pairs with planned payloads back into the algorithms presented in this dissertation for enhanced route optimization.

Appendix A: Nodal Reduction and Route Generation Algorithms

primaryAirfield Object

Property	Description
id	Identification.
distFromPrim	Distance in NMs from origin airfield to primary airfield.
azFromPrim	Azimuth in degrees from origin airfield to primary airfield.
altFromPrim	Altitude in thousands of feet from origin airfield to primary airfield.
distPrimTo	Distance in NMs from primary airfield to destination airfield.
azPrimTo	Azimuth in degrees from primary airfield to destination airfield.
altPrimTo	Altitude in thousands of feet from primary airfield to destination airfield.
totalDist	Total distance from origin to primary to destination in NMs.
maxPCNgw	Maximum aircraft gross weight for given PCN.

```
// function that creates the primaryAirfield() object
function primaryAirfield(id, distFromPrim, azFromPrim, altFromPrim, distPrimTo, azPrimTo, altPrimTo,
                         totalDist, maxPCNgw) {
    this.id = id;
    this.distFromPrim = distFromPrim;
    this.azFromPrim = azFromPrim;
    this.altFromPrim = altFromPrim;
    this.distPrimTo = distPrimTo;
    this.azPrimTo = azPrimTo;
    this.altPrimTo = altPrimTo;
    this.totalDist = totalDist;
    this.maxPCNgw = maxPCNgw;
}
```

secondaryAirfield Object

Property	Description
id	Identification.
selAfldsPos	The id of the primary airfield that is the same airfield as the secondary.
distPrimSec	Distance in NMs from primary to secondary airfield.
azPrimSec	Azimuth in degrees from primary to secondary airfield.
altPrimSec	Altitude in thousands of feet from primary to secondary airfield.
distSecTo	Distance in NMs from secondary airfield to destination airfield.
azSecTo	Azimuth in degrees from secondary airfield to destination airfield.
altSecTo	Altitude in thousands of feet from secondary airfield to destination airfield.
totalDist	Total distance from primary to secondary to destination in NMs.

```
// function that creates the secondaryAirfield() object
function secondaryAirfield(id, selAfldsPos, distPrimSec, azPrimSec, altPrimSec, distSecTo, azSecTo,
                           altSecTo, totalDist) {
    this.id = id;
    this.selAfldsPos = selAfldsPos;
    this.distPrimSec = distPrimSec;
    this.azPrimSec = azPrimSec;
    this.altPrimSec = altPrimSec;
    this.distSecTo = distSecTo;
    this.azSecTo = azSecTo;
    this.altSecTo = altSecTo;
    this.totalDist = totalDist;
}
```

route Object

Property	Description
sortieArray	Array of sortie objects
numStops	Number of stops along the route.
totalDist	Total distance along the route in NMs.
deliveryTime	Delivery time from crew show at origin to landing at destination in hours.
cycleTime	Cycle time from takeoff at origin to next takeoff at origin in hours.
maxPayloadNoTrans	Maximum payload without transload operations which is the minimum of all the maximum payloads in the sortieArray object.
cargoThroughput	Cargo throughput in Klbs of cargo per day.
fuelEfficiency	Fuel efficiency which is cargo throughput per day over fuel consumed per day.
cargoThruPlanPay	Cargo throughput in Klbs of cargo per day for the planned payload.
fuelEffPlanPay	Fuel efficiency which is cargo throughput for the planned payload per day over fuel consumed for the planned payload per day.

```
// function that creates the route() object
function route(sortieArray, numStops, totalDist, deliveryTime, cycleTime, maxPayloadNoTrans,
cargoThroughput, fuelEfficiency, cargoThruPlanPay, fuelEffPlanPay) {
    this.sortieArray = sortieArray;
    this.numStops = numStops;
    this.totalDist = totalDist;
    this.deliveryTime = deliveryTime;
    this.cycleTime = cycleTime;
    this.maxPayloadNoTrans = maxPayloadNoTrans;
    this.cargoThroughput = cargoThroughput;
    this.fuelEfficiency = fuelEfficiency;
    this.cargoThruPlanPay = cargoThruPlanPay;
    this.fuelEffPlanPay = fuelEffPlanPay;
}
```

sortie Object

Property	Description
fromIdent	Identity of the origin.
toIdent	Identity of the destination
distance	Total distance in NMs for that sortie from origin to destination.
cargoThroughput	Cargo throughput in Klbs of cargo per day for that sortie.
enrouteTime	Time from takeoff to landing in hours.
enrouteFuel	Total fuel from takeoff to landing in Klbs.
sortieParam	An instance of the sortieParam object from the AircraftMDS.js code.

```
// function that creates the sortie() object
function sortie(fromIdent, toIdent, distance, cargoThroughput, enrouteTime, enrouteFuel, sortieParam) {
    this.fromIdent = fromIdent;
    this.toIdent = toIdent;
    this.distance = distance;
    this.cargoThroughput = cargoThroughput;
    this.enrouteTime = enrouteTime;
    this.enrouteFuel = enrouteFuel;
    this.sortieParam = sortieParam;
}
```

Route Functions

Name	Inputs	Description
DetermineProgress	()	Displays the progress bar.
AlterAircraft FilterParameters	()	When MDS dropdown is changed, adjusts parameters in their respective input boxes to selected MDS appropriate values.
SortRoutes	()	Sorts the routes based on the parameter selected and displays the information.
UpdateRteString1	(rteString)	Places the selected route string in the first route for comparison.
UpdateRteString2	(rteString)	Places the selected route string in the second route for comparison.
CreateIranPoly	()	Creates the polygon for Iranian restricted airspace.
CreateRussiaPoly	()	Creates the polygon for Russian restricted airspace.
AddWaypointToPoly	()	Adds waypoint to restricted airspace polygon.
ClearPoly	()	Clears the restricted airspace polygon of all waypoints.
ValidateRouteInput	()	Validates all input fields.
FilterAirfields ComputePrimSec AndBuildRoutes	()	Calls the AirfieldNodalReduction function to filter airfields and select the set of primary airfields. Calls the DetermineSecondaryAirfields function to calculate the set of secondary airfields for each primary. Calls either the BuildRoutes or BuildRoutesRestAS functions to build the proper routes.
AirfieldNodal Reduction	(selectMDS, topNumAflds)	Utilizes the 2D Array selectedAirfields to store all primaryAirfield objects in the 0 position of the set of arrays. The primary airfields are culled from all airfields using country, effective runway length, runway width, pavement strength, cutoff distance and finally total distance multiple filters
IsPavement StrengthAnomaly	(selectMDS, afldID, distFromEnr, distFromTo)	Returns boolean true if the pavement strength of the selected airfield is more than 10% stronger than origin airfield and the airfield is within minimum cutoff distance.
HasRunwayThatIs LongWideAnd StrongEnough	(selectMDS, airfield, minEffRwyLen, gwCFLave, gwPaveMin)	Returns boolean true if the airfield has a runway that meets runway length, width and pavement strength requirements.
EffectiveRunway Length	(selectMDS, gw, actRwyLength, latitude, elevation)	Returns the effective runway length in feetgiven aircraft gross weight, latitude and elevation.
IsCountryRemove Airfield	(afld)	Returns boolean true is airfield is in one of the countries selected for removal.
IsCutDistMod RemoveAirfield	(selectMDS, distFromEnr, distEnrTo, distFromTo)	Returns boolean true if airfield is within cutoff distance of origin or destination.

Name	Inputs	Description
DetermineSecondaryAirfields	(selectMDS, toIdent, topNumSecAflds)	Utilizes the 2D Array selectedAirfields to store all secondaryAirfield objects associated with the primary airfields in the 0 position of the set of arrays.
ClearSortBox	(sortParam)	Treats the sort check boxes like radio buttons so that only one can be selected at a given time.
DisplayInput	()	Toggles the display of the input section so the results can be more easily seen.
BuildRoutes	(selectMDS, fromIdent, toIdent, maxNumStops, planPay, augmented, staging, transload, numStopForRoute, numPrimary)	Builds an Array of route objects that consists of an array of sortie objects. This set of routes represents all potential route combinations for the set of primary and associated secondary airfields.
BuildRoutesRestAS	(selectMDS, fromIdent, toIdent, maxNumStops, planPay, augmented, staging, transload, numStopForRoute, numPrimary)	Builds an Array of route objects that consists of an array of sortie objects. This set of routes represents all potential route combinations for the set of primary and associated secondary airfields. Allows for the sortie to go around a restricted airspace.

```
// global variables
var initSelectAflds = new Array();// initial set of selected airfields

// creates two dimensional array with first position in each row representing primary airfields.
// additional columns of that row represent secondary airfields to the primary
var selectedAirfields = new Array(new Array());
var progressComplete, currentProgress1, currentProgress2;
var progressArray = [{ category: "Calculating Primary and Secondary Airfields",
                    percent: 0 }, { category: "Building Routes", percent: 0}];
var numStopForRoute, numPrimary, globalFromIdent, globalToIdent;
var routes = new Array();

// function called for calculation after input data validated.  Measures and displays progress
function DetermineProgress() {
    var bar = document.getElementById('progressBar');
    var meter = document.getElementById('meterID');
    var progressMessage = document.getElementById('progressMessage');
    var sortDiv = document.getElementById("sortDiv");
    var routeSummaryDiv = document.getElementById("routeSummaryDiv");
    var routeResultDiv = document.getElementById("routeResultDiv");
    var debugDiv = document.getElementById("debugDiv");

    sortDiv.style.display = "none";
    routeSummaryDiv.style.display = "none";
    routeResultDiv.style.display = "none";

    if (progressArray[0].percent < 100) {
        progressMessage.innerHTML = progressArray[0].category;
        progressMessage.style.display = "inline-block";
    }
    else if (progressArray[0].percent == 100) {
        progressMessage.innerHTML = progressArray[1].category;
        progressMessage.style.display = "inline-block";
        bar.style.width = (10 * progressArray[1].percent) + "px";
        meter.style.display = "inline-block";
    }
    FilterAirfieldsComputePrimSecAndBuildRoutes();
```

```
    var inProgressTimer = setTimeout("DetermineProgress()", 10);
    if (progressComplete) {
        clearTimeout(inProgressTimer);
        meter.style.display = "none";
        progressMessage.style.display = "none";
    }
}

// function called from DetermineProgress that performs 3 primary functions
// 1-filters airfields and determines primary airfields using AirfieldNodalReduction() function
// 2-obtains secondary airfields using DetermineSecondaryAirfields() function
// 2-builds routes using BuildRoutes() function
function FilterAirfieldsComputePrimSecAndBuildRoutes() {
    var mds = document.getElementById("MDS").value;
    var minEffRwyLen = parseFloat(document.getElementById("minEffRwyLen").value);
    var gwCFLave = parseFloat(document.getElementById("gwCFLave").value);
    var gwPaveMin = parseFloat(document.getElementById("gwPaveMin").value);
    var plannedPayload = parseInt(document.getElementById("plannedPayload").value);
    var fromICAO = document.getElementById("searchICAO1").value.toUpperCase();
    var toICAO = document.getElementById("searchICAO2").value.toUpperCase();
    var topNumAflds = parseInt(document.getElementById("topNumAflds").value);
    var topNumSecAflds = parseInt(document.getElementById("topNumSecAflds").value);
    var maxNumLegs = parseInt(document.getElementById("maxNumLegs").value);
    var augmentedCheck = document.getElementById("augmentedCheck").checked;
    var stagingCheck = document.getElementById("stagingCheck").checked;
    var transloadCheck = document.getElementById("transloadCheck").checked;
    var progress2iteration, progress2itMax;
    var selectMDS;
    progressComplete = false;
    switch (mds) {
        case "1":
            selectMDS = C5;
            break;
        case "2":
            selectMDS = C17;
            break;
        case "3":
            selectMDS = C130;
            break;
        default:
            break;
    }
    if (progressArray[0].percent < 100) {
        AirfieldNodalReduction(selectMDS, minEffRwyLen, gwCFLave, gwPaveMin, globalFromIdent,
                               globalToIdent, topNumAflds);
        DetermineSecondaryAirfields(selectMDS, globalToIdent, topNumSecAflds);
        routes = [];
        currentProgress1 = 100;
        currentProgress2 = 0;
        numStopForRoute = 0;
        numPrimary = 0;
    }
    else if (progressArray[0].percent >= 100 && progressArray[1].percent < 100) {
        BuildRoutes(selectMDS, globalFromIdent, globalToIdent, maxNumLegs - 1, plannedPayload,
                    augmentedCheck, stagingCheck, transloadCheck,
                    numStopForRoute, numPrimary);
        if (numStopForRoute == 0)
            numStopForRoute++;
        else if (numPrimary == (selectedAirfields.length - 1)) {
            numStopForRoute++;
            numPrimary = 0;
        }
        else {
            numPrimary++;
        }
        progress2iteration = (numStopForRoute - 1) * selectedAirfields.length + numPrimary;
        progress2itMax = (maxNumLegs - 1) * selectedAirfields.length;
        if (selectedAirfields.length != 0)
            currentProgress2 = parseInt(Math.round((progress2iteration / progress2itMax) * 100));
```

```
        else
            currentProgress2 = 100;
    }
    else if (progressArray[0].percent >= 100 && progressArray[1].percent >= 100) {
        SortRoutes();
        progressComplete = true;
        currentProgress1 = 0;
        currentProgress2 = 0;

    }
    progressArray[0].percent = currentProgress1;
    progressArray[1].percent = currentProgress2;
}

// function that reduces the number of airfields under consideration using runway length, width, pavement
// strength and minimum cutoff distance
function AirfieldNodalReduction(selectMDS, minEffRwyLen, gwCFLave, gwPaveMin, fromIdent, toIdent,
topNumAflds) {
    var selectThisAirfield = true;
    var GeoCurveFromEnr = 0.0;
    var GeoCurveEnrTo = 0.0;
    var distFromEnr = 0.0;
    var distEnrTo = 0.0;
    var altFromPrim = 0.0;
    var altPrimTo = 0.0;
    var maxPCNtakeoffWeight;
    var GeoCurve = VincentyDistance(airfields[fromIdent].wgs_dlat, airfields[fromIdent].wgs_dlong,
                                    airfields[toIdent].wgs_dlat, airfields[toIdent].wgs_dlong);
    var distFromTo = GeoCurve.distance / 1852.0;
    var selectedCount = 0;

    initSelectAflds = [];
    selectedAirfields = [];

    for (var i = 0; i < airfields.length; i++) {
        selectThisAirfield = true;
        //Remove airfields that are in countries selected by the user
        if (IsCountryRemoveAirfield(airfields[i]))
            selectThisAirfield = false;

        //Remove airfield without a runway that meets both the greater of minimum runway length and
        //effective runway length and runway width criteria and pavement criteria
        if (!HasRunwayThatIsLongWideAndStrongEnough(selectMDS, airfields[i], minEffRwyLen, gwCFLave,
            gwPaveMin)) {
            selectThisAirfield = false;
        }
        if (selectThisAirfield)
            initSelectAflds.push(i);
    }

    for (var j = 0; j < initSelectAflds.length; j++) {
        selectThisAirfield = true;

        // determine distances from source to en route and en route to destination
        GeoCurveFromEnr = VincentyDistance(airfields[fromIdent].wgs_dlat, airfields[fromIdent].wgs_dlong,
                                           airfields[initSelectAflds[j]].wgs_dlat,
                                           airfields[initSelectAflds[j]].wgs_dlong);
        GeoCurveEnrTo = VincentyDistance(airfields[initSelectAflds[j]].wgs_dlat,
                                         airfields[initSelectAflds[j]].wgs_dlong,
                                         airfields[toIdent].wgs_dlat, airfields[toIdent].wgs_dlong);
        distFromEnr = GeoCurveFromEnr.distance / 1852.0;
        distEnrTo = GeoCurveEnrTo.distance / 1852.0;

        //Remove airfields that are not in eye shape formed by minimum cutoff distance model
        if (IsCutDistModRemoveAirfield(selectMDS, distFromEnr, distEnrTo, distFromTo))
            selectThisAirfield = false;

        // add fields to selected airfields
        if (selectThisAirfield) {
```

```
            altFromPrim = selectMDS.OptimumAltitudeMaxGW(distFromEnr, GeoCurveFromEnr.azimuth);
            altPrimTo = selectMDS.OptimumAltitudeMaxGW(distEnrTo, GeoCurveEnrTo.azimuth);
            selectedAirfields.push([]);
            selectedAirfields[selectedCount].push(new primaryAirfield(initSelectAflds[j], distFromEnr,
                                        GeoCurveFromEnr.azimuth, altFromPrim, distEnrTo,
                                        GeoCurveEnrTo.azimuth, altPrimTo, distFromEnr + distEnrTo, 0));
            selectedCount++;
        }
    }

    // sort airfields by total distance
    selectedAirfields = selectedAirfields.sort(function (a, b) {
        if (a[0].totalDist > b[0].totalDist)
            return 1;
        else
            return -1;
    });

    // select only the top number of airfields suggested by the user
    if (selectedAirfields.length > topNumAflds) {
        selectedAirfields.splice(topNumAflds, selectedAirfields.length - topNumAflds);
    }

    for (var k = 0; k < selectedAirfields.length; k++) {
        maxPCNtakeoffWeight = 0;
        for (var m = 0; m < airfields[selectedAirfields[k][0].id].runways.length; m++) {
            PCNtakeoffWeight = PCNtoMaxGrossWeight(selectMDS,
                                                   airfields[selectedAirfields[k][0].id].runways[m].pcn);
            if (PCNtakeoffWeight > maxPCNtakeoffWeight) {
                maxPCNtakeoffWeight = PCNtakeoffWeight;
            }
        }
        selectedAirfields[k][0].maxPCNgw = maxPCNtakeoffWeight;
    }
}

// function that returns a Boolean on whether a runway exceeds minimum length and width rqmts for MDS
function HasRunwayThatIsLongWideAndStrongEnough(selectMDS, afld, minEffRwyLen, gwCFLave, gwPaveMin) {
    var isLongWideAndStrongEnough = false;
    var acftRwyLenMin = 0;
    var acftRwyWidMin = 0;

    acftRwyLenMin = selectMDS.minRwyLength;
    acftRwyWidMin = selectMDS.minRwyWidth;

    for (var i = 0; i < afld.runways.length; i++) {
        if (afld.runways[i].rwyLength >= acftRwyLenMin && afld.runways[i].width > acftRwyWidMin) {
            if (EffectiveRunwayLength(selectMDS, gwCFLave, afld.runways[i].rwyLength, afld.wgs_dlat,
                afld.elev) > minEffRwyLen) {
                if (PCNtoMaxGrossWeight(selectMDS, afld.runways[i].pcn) > gwPaveMin)
                    isLongWideAndStrongEnough = true;
            }
        }
    }
    return isLongWideAndStrongEnough;
}

// function that returns effective runway length
function EffectiveRunwayLength(selectMDS, gw, actRwyLength, latitude, elevation)
{
    var seaLevelStdDayCFL;
    var airfieldCFL;
    var effRwyLength;
    var airfieldTemp;
    var seaLevelElev = 0.0;
    var stdDayTemp = 15.0;

    //Calculate the airfields temperature
    airfieldTemp = DetermineAverageTemperature(latitude, elevation);
```

```
    //Calculate the critical field lengths at sea level std day and at the airfield
    seaLevelStdDayCFL = selectMDS.CriticalFieldLength(gw, seaLevelElev, stdDayTemp);
    airfieldCFL = selectMDS.CriticalFieldLength(gw, elevation / 1000.0, airfieldTemp);

    //Calculate the effective runway length
    effRwyLength = actRwyLength * seaLevelStdDayCFL / airfieldCFL;

    return effRwyLength;
}

// function that removes a given airfield if it is in a country that is filtered out
function IsCountryRemoveAirfield(afld) {
    var removeAfld = false;

    var russiaCheck = document.getElementById("russiaCheck");
    var chinaCheck = document.getElementById("chinaCheck");
    var venezuelaCheck = document.getElementById("venezuelaCheck");
    var iranCheck = document.getElementById("iranCheck");

    if (russiaCheck.checked)
        if (afld.icao.substring(0, 2) == "UE" ||
            afld.icao.substring(0, 2) == "UH" ||
            afld.icao.substring(0, 2) == "UI" ||
            afld.icao.substring(0, 2) == "UL" ||
            afld.icao.substring(0, 2) == "UN" ||
            afld.icao.substring(0, 2) == "UO" ||
            afld.icao.substring(0, 2) == "UR" ||
            afld.icao.substring(0, 2) == "US" ||
            afld.icao.substring(0, 2) == "UU" ||
            afld.icao.substring(0, 2) == "UW")
            removeAfld = true;

    if (chinaCheck.checked)
        if (afld.icao.substring(0, 1) == "Z")
            removeAfld = true;

    if (venezuelaCheck.checked)
        if (afld.icao.substring(0, 2) == "SV")
            removeAfld = true;

    if (iranCheck.checked)
        if (afld.icao.substring(0, 2) == "OI")
            removeAfld = true;

    return removeAfld;
}

// function that suggests to remove airfield if it is belowe the cutoff distance
function IsCutDistModRemoveAirfield(selectMDS, distFromEnr, distEnrTo, distFromTo) {
    var removeAfld = false;
    var cutoffDist = 0.0;

    switch (selectMDS) {
        case C5:
            if (distFromTo <= 2300)
                cutoffDist = distFromTo;
            else if (distFromTo > 2300 && distFromTo < 3300)
                cutoffDist = 800;
            else
                cutoffDist = 400;
            break;
        case C17:
            if (distFromTo <= 2200)
                cutoffDist = distFromTo;
            else if (distFromTo > 2200 && distFromTo < 3500)
                cutoffDist = 800;
            else
                cutoffDist = 400;
```

```
            break;
        case C130:
            if (distFromTo <= 1700)
                cutoffDist = distFromTo;
            else if (distFromTo > 1700 && distFromTo < 2700)
                cutoffDist = 600;
            else
                cutoffDist = 400;
            break;
        default:
            break;
    }

    if (distFromEnr > (distFromTo - cutoffDist) || distEnrTo > (distFromTo - cutoffDist))
        removeAfld = true;

    return removeAfld;
}

// function that determines the secondary airfields
function DetermineSecondaryAirfields(selectMDS, toIdent, topNumSecAflds) {
    var secondaryAirfields = new Array();
    var GeoCurvePrimSec;
    var distPrimSec = 0.0;
    var altPrimSec = 0.0;
    var altSecTo = 0.0;

    //sort primary airfields by distance to destination
    selectedAirfields = selectedAirfields.sort(function (a, b) {
        if (a[0].distPrimTo > b[0].distPrimTo)
            return -1;
        else
            return 1;
    });

    // loop through primary airfields
    for (var i = 0; i < selectedAirfields.length; i++) {
        secondaryAirfields = new Array();
        if (topNumSecAflds > 0) {
            // loop through potential secondary airfields
            for (var j = i + 1; j < selectedAirfields.length; j++) {
                //determine distance from primary to secondary
                GeoCurvePrimSec = VincentyDistance(airfields[selectedAirfields[i][0].id].wgs_dlat,
                                                   airfields[selectedAirfields[i][0].id].wgs_dlong,
                                                   airfields[selectedAirfields[j][0].id].wgs_dlat,
                                                   airfields[selectedAirfields[j][0].id].wgs_dlong);
                distPrimSec = GeoCurvePrimSec.distance / 1852.0;

                //determine distance from secondary to destination
                GeoCurveSecTo = VincentyDistance(airfields[selectedAirfields[j][0].id].wgs_dlat,
                                                 airfields[selectedAirfields[j][0].id].wgs_dlong,
                                                 airfields[toIdent].wgs_dlat,
                                                 airfields[toIdent].wgs_dlong);
                distSecTo = GeoCurveSecTo.distance / 1852.0;

                //determine if within eye shape formed by cutoff distance and add to set of potential
                if (!IsCutDistModRemoveAirfield(selectMDS, distPrimSec, distSecTo,
                                                selectedAirfields[i][0].distPrimTo)) {
                    altPrimSec = selectMDS.OptimumAltitudeMaxGW(distPrimSec, GeoCurvePrimSec.azimuth);
                    altSecTo = selectMDS.OptimumAltitudeMaxGW(distSecTo, GeoCurveSecTo.azimuth);
                    secondaryAirfields.push(new secondaryAirfield(selectedAirfields[j][0].id, j,
                                                                  distPrimSec, GeoCurvePrimSec.azimuth, altPrimSec,
                                                                  distSecTo, GeoCurveSecTo.azimuth, altSecTo,
                                                                  distPrimSec + distSecTo));
                }
            }
        }

        // sort secondary airfields by total distance
```

```
        secondaryAirfields.sort(function (a, b) {
            if (a.totalDist > b.totalDist)
                return 1;
            else
                return -1;
        });

        // add only max number of secondary
        if (secondaryAirfields.length > topNumSecAflds) {
            secondaryAirfields.splice(topNumSecAflds, secondaryAirfields.length - topNumSecAflds);
        }
        selectedAirfields[i][1] = secondaryAirfields;
    }
}

// function that builds routes
function BuildRoutes(selectMDS, fromIdent, toIdent, maxNumStops, planPay, augmented, staging, transload,
                     numStopForRoute, numPrimary) {
    var climbAWF = 0;
    var enrouteAWF = 0;
    var enrouteDeltaT = 0;
    var descendAWF = 0;
    var maxPayloadNoTrans = 0;
    var GeoCurveFromTo;
    var distFromTo = 0;

    var sortie1Param, sortie2Param, sortie3Param, sortie4Param, sortie5Param;
    var PCNtakeoffWeight = 0;
    var maxPCNtakeoffWeight;
    var totalSortie1Time, totalSortie2Time, totalSortie3Time, totalSortie4Time, totalSortie5Time;
    var totalSortie1Fuel, totalSortie2Fuel, totalSortie3Fuel, totalSortie4Fuel, totalSortie5Fuel;
    var routeFuelNoTrans, routeFuelTrans, routeFuelTransPlanPay;
    var cargoThruTransSrt1, cargoThruTransSrt2, cargoThruTransSrt3, cargoThruTransSrt4,
        cargoThruTransSrt5;
    var cargoThruTransPlanPaySrt1, cargoThruTransPlanPaySrt2, cargoThruTransPlanPaySrt3,
        cargoThruTransPlanPaySrt4, cargoThruTransPlanPaySrt5;
    var deliveryTime = 0;
    var cycleTime = 0;
    var totalDistance = 0;
    var altLeg1, altLeg2, altLeg3, altLeg4, altLeg5;
    var srt1, srt2, srt3, srt4, srt5;
    var srtArr = new Array();
    var k, m, p, s;
    var maxTransCargoThru, maxTransCargoThruPlanPay;

    k = numPrimary;
    m = numPrimary;
    p = numPrimary;
    s = numPrimary;

    // determine maxPCNweight
    maxPCNtakeoffWeight = 0;
    for (var j = 0; j < airfields[fromIdent].runways.length; j++) {
        PCNtakeoffWeight = PCNtoMaxGrossWeight(selectMDS, airfields[fromIdent].runways[j].pcn);
        if (PCNtakeoffWeight > maxPCNtakeoffWeight)
            maxPCNtakeoffWeight = PCNtakeoffWeight;
    }

    // determine distance origin to destination
    GeoCurveFromTo = VincentyDistance(airfields[fromIdent].wgs_dlat, airfields[fromIdent].wgs_dlong,
                                      airfields[toIdent].wgs_dlat, airfields[toIdent].wgs_dlong);
    distFromTo = GeoCurveFromTo.distance / 1852.0;

    // determine optimal altitude
    altLeg1 = selectMDS.OptimumAltitudeMaxGW(distFromTo, GeoCurveFromTo.azimuth);

    var maxRange = selectMDS.MaxDistanceForSortie(maxPCNtakeoffWeight, 0, 0, 0, 0, 0, 0, enrouteDeltaT,
                                                  altLeg1);
```

```
if (numStopForRoute == 0) {
    // is the sortie possible zero payload
    if (distFromTo <= maxRange) {
        // determine sortie parameters
        sortie1Param = selectMDS.FuelConsumedIterationGrossWeightFixed(maxPCNtakeoffWeight, altLeg1,
                        distFromTo, airfields[fromIdent].elev / 1000, airfields[toIdent].elev / 1000,
                        climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);

        // create sortie
        totalSortie1Time = sortie1Param.TotalTime();
        totalSortie1Fuel = sortie1Param.TotalFuel();

        if (transload) {
            cargoThruTransSrt1 = selectMDS.CargoThroughputTransload(totalSortie1Time,
                                                                    sortie1Param.payload, augmented, staging);
            if (planPay > sortie1Param.payload)
                cargoThruTransPlanPaySrt1 = cargoThruTransSrt1;
            else
                cargoThruTransPlanPaySrt1 = selectMDS.CargoThroughputTransload(totalSortie1Time,
                                                                               planPay, augmented, staging);
        }
        else
            cargoThruTransSrt1 = 0.0;

        srt1 = new sortie(fromIdent, toIdent, distFromTo, cargoThruTransSrt1, totalSortie1Time,
                          totalSortie1Fuel, sortie1Param);

        // add sortie to Array
        srtArr = []
        srtArr.push(srt1);

        // create route and add sortie to route
        var rte = new route(srtArr, 0, srt1.distance, 0, 0, sortie1Param.payload, 0, 0, 0, 0);

        // determine delivery and cycle times for the route
        rte.deliveryTime = selectMDS.RouteDeliveryTimeCalculator(srtArr, augmented, transload,
                                                                 staging);
        rte.cycleTime = selectMDS.RouteCycleTimeCalculator(srtArr, augmented, transload, staging);

        // determine cargo throughput
        maxTransCargoThru = srt1.cargoThroughput;
        maxTransCargoThruPlanPay = cargoThruTransPlanPaySrt1;
        if (transload) {
            rte.cargoThroughput = srt1.cargoThroughput;
            rte.cargoThruPlanPay = cargoThruTransPlanPaySrt1;
        }
        else {
            rte.cargoThroughput = 24 * rte.maxPayloadNoTrans / rte.cycleTime;
            if (planPay > rte.maxPayloadNoTrans)
                rte.cargoThruPlanPay = rte.cargoThroughput;
            else
                rte.cargoThruPlanPay = 24 * planPay / rte.cycleTime;
        }

        // determine fuel efficiency
        routeFuelTrans = selectMDS.DailyFuelConsumptionTrans(srtArr, augmented, staging,
                                                             maxTransCargoThru);
        routeFuelTransPlanPay = selectMDS.DailyFuelConsumptionTrans(srtArr, augmented, staging,
                                                                    maxTransCargoThruPlanPay);
        if (transload) {
            rte.fuelEfficiency = rte.cargoThroughput / routeFuelTrans;
            rte.fuelEffPlanPay = rte.cargoThruPlanPay / routeFuelTransPlanPay;
        }
        else {
            rte.fuelEfficiency = rte.cargoThroughput / (24 * 2 * totalSortie1Fuel / rte.cycleTime);
            rte.fuelEffPlanPay = rte.cargoThruPlanPay / (24 * 2 * totalSortie1Fuel / rte.cycleTime);
        }

        // add route to Array of routes
```

```
            routes.push(rte);
        }
    }
    else if (numStopForRoute == 1) {
        // is the first sortie possible zero payload
        if (selectedAirfields[k][0].distFromPrim <= maxRange) {
            // is the second sortie possible zero payload
            if (selectedAirfields[k][0].distPrimTo <= maxRange) {
                // determine sortie parameters
                sortie1Param = selectMDS.FuelConsumedIterationGrossWeightFixed(maxPCNtakeoffWeight,
                     selectedAirfields[k][0].altFromPrim, selectedAirfields[k][0].distFromPrim,
                     airfields[fromIdent].elev / 1000, airfields[selectedAirfields[k][0].id].elev / 1000,
                     climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
                sortie2Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
                                    selectedAirfields[k][0].maxPCNgw, selectedAirfields[k][0].altPrimTo,
                                    selectedAirfields[k][0].distPrimTo,
                                    airfields[selectedAirfields[k][0].id].elev / 1000,
                                    airfields[toIdent].elev / 1000,
                                    climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);

                // create sortie
                totalSortie1Time = sortie1Param.TotalTime();
                totalSortie1Fuel = sortie1Param.TotalFuel();
                totalSortie2Time = sortie2Param.TotalTime();
                totalSortie2Fuel = sortie2Param.TotalFuel();

                if (transload) {
                    cargoThruTransSrt1 = selectMDS.CargoThroughputTransload(totalSortie1Time,
                                                        sortie1Param.payload, augmented, staging);
                    cargoThruTransSrt2 = selectMDS.CargoThroughputTransload(totalSortie2Time,
                                                        sortie2Param.payload, augmented, staging);
                    if (planPay > sortie1Param.payload)
                        cargoThruTransPlanPaySrt1 = cargoThruTransSrt1;
                    else
                        cargoThruTransPlanPaySrt1 = selectMDS.CargoThroughputTransload(totalSortie1Time,
                                                                    planPay, augmented, staging);
                    if (planPay > sortie2Param.payload)
                        cargoThruTransPlanPaySrt2 = cargoThruTransSrt2;
                    else
                        cargoThruTransPlanPaySrt2 = selectMDS.CargoThroughputTransload(totalSortie2Time,
                                                                    planPay, augmented, staging);
                }
                else {
                    cargoThruTransSrt1 = 0.0;
                    cargoThruTransSrt2 = 0.0;
                }

                srt1 = new sortie(fromIdent, selectedAirfields[k][0].id,
                                  selectedAirfields[k][0].distFromPrim, cargoThruTransSrt1,
                                  totalSortie1Time, totalSortie1Fuel, sortie1Param);
                srt2 = new sortie(selectedAirfields[k][0].id, toIdent, selectedAirfields[k][0].distPrimTo,
                                  cargoThruTransSrt2, totalSortie2Time, totalSortie2Fuel, sortie2Param);

                // add sortie to Array
                srtArr = [];
                srtArr.push(srt1);
                srtArr.push(srt2);

                // create route and add sortie to route
                totalDistance = srt1.distance + srt2.distance;
                maxPayloadNoTrans = Math.min(sortie1Param.payload, sortie2Param.payload);
                routeFuelNoTrans = srt1.enrouteFuel + srt2.enrouteFuel;
                var rte = new route(srtArr, 1, totalDistance, 0, 0, maxPayloadNoTrans, 0, 0, 0, 0);

                // determine delivery and cycle times for the route
                rte.deliveryTime = selectMDS.RouteDeliveryTimeCalculator(srtArr, augmented, transload,
                                                                         staging);
                rte.cycleTime = selectMDS.RouteCycleTimeCalculator(srtArr, augmented, transload, staging);
```

```
            // determine cargo throughput
            maxTransCargoThru = Math.max(srt1.cargoThroughput, srt2.cargoThroughput);
            maxTransCargoThruPlanPay = Math.max(cargoThruTransPlanPaySrt1, cargoThruTransPlanPaySrt2);
            if (transload) {
                rte.cargoThroughput = maxTransCargoThru /
                                      ((maxTransCargoThru / srt1.cargoThroughput) +
                                       (maxTransCargoThru / srt2.cargoThroughput));
                rte.cargoThruPlanPay = maxTransCargoThruPlanPay /
                                       ((maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt1) +
                                       (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt2));
            }
            else {
                rte.cargoThroughput = 24 * rte.maxPayloadNoTrans / rte.cycleTime;
                if (planPay > rte.maxPayloadNoTrans)
                    rte.cargoThruPlanPay = rte.cargoThroughput;
                else
                    rte.cargoThruPlanPay = 24 * planPay / rte.cycleTime;
            }

            // determine fuel efficiency
            routeFuelTrans = selectMDS.DailyFuelConsumptionTrans(srtArr, augmented, staging,
                                                                 maxTransCargoThru);
            routeFuelTransPlanPay = selectMDS.DailyFuelConsumptionTrans(srtArr, augmented, staging,
                                                                        maxTransCargoThruPlanPay);
            if (transload) {
                rte.fuelEfficiency = rte.cargoThroughput / routeFuelTrans;
                rte.fuelEffPlanPay = rte.cargoThruPlanPay / routeFuelTransPlanPay;
            }
            else {
                rte.fuelEfficiency = rte.cargoThroughput /
                                     (24 * 2 * routeFuelNoTrans / rte.cycleTime);
                rte.fuelEffPlanPay = rte.cargoThruPlanPay /
                                     (24 * 2 * routeFuelNoTrans / rte.cycleTime);
            }

            // add route to Array of routes
            routes.push(rte);
        }
    }
}
else if (numStopForRoute == 2) {
    // determine if a secondary airfield exists and the first sortie is possible zero payload
    if (selectedAirfields[m][1].length != 0 && (selectedAirfields[m][0].distFromPrim < maxRange)) {
        // loop through the secondary airfields
        for (var n = 0; n < selectedAirfields[m][1].length; n++) {
            // is the second sortie possible zero payload
            if (selectedAirfields[m][1][n].distPrimSec <= maxRange) {
                // is the third sortie possible zero payload
                if (selectedAirfields[m][1][n].distSecTo <= maxRange) {
                    // determine sortie parameters
                    sortie1Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
                                            maxPCNtakeoffWeight,
                                            selectedAirfields[m][0].altFromPrim,
                                            selectedAirfields[m][0].distFromPrim,
                                            airfields[fromIdent].elev / 1000,
                                            airfields[selectedAirfields[m][0].id].elev / 1000,
                                            climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
                    sortie2Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
                                            selectedAirfields[m][0].maxPCNgw,
                                            selectedAirfields[m][1][n].altPrimSec,
                                            selectedAirfields[m][1][n].distPrimSec,
                                            airfields[selectedAirfields[m][0].id].elev / 1000,
                                            airfields[selectedAirfields[m][1][n].id].elev / 1000,
                                            climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
                    sortie3Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
                              selectedAirfields[selectedAirfields[m][1][n].selAfldsPos][0].maxPCNgw,
                              selectedAirfields[m][1][n].altSecTo,
                              selectedAirfields[m][1][n].distSecTo,
```

```
            airfields[selectedAirfields[m][1][n].id].elev / 1000,
            airfields[toIdent].elev / 1000,
            climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);

// create sortie
totalSortie1Time = sortie1Param.TotalTime();
totalSortie1Fuel = sortie1Param.TotalFuel();
totalSortie2Time = sortie2Param.TotalTime();
totalSortie2Fuel = sortie2Param.TotalFuel();
totalSortie3Time = sortie3Param.TotalTime();
totalSortie3Fuel = sortie3Param.TotalFuel();

if (transload) {
    cargoThruTransSrt1 = selectMDS.CargoThroughputTransload(totalSortie1Time,
                                    sortie1Param.payload, augmented, staging);
    cargoThruTransSrt2 = selectMDS.CargoThroughputTransload(totalSortie2Time,
                                    sortie2Param.payload, augmented, staging);
    cargoThruTransSrt3 = selectMDS.CargoThroughputTransload(totalSortie3Time,
                                    sortie3Param.payload, augmented, staging);
    if (planPay > sortie1Param.payload)
        cargoThruTransPlanPaySrt1 = cargoThruTransSrt1;
    else
        cargoThruTransPlanPaySrt1 = selectMDS.CargoThroughputTransload(
                                totalSortie1Time, planPay, augmented, staging);
    if (planPay > sortie2Param.payload)
        cargoThruTransPlanPaySrt2 = cargoThruTransSrt2;
    else
        cargoThruTransPlanPaySrt2 = selectMDS.CargoThroughputTransload(
                                totalSortie2Time, planPay, augmented, staging);
    if (planPay > sortie3Param.payload)
        cargoThruTransPlanPaySrt3 = cargoThruTransSrt3;
    else
        cargoThruTransPlanPaySrt3 = selectMDS.CargoThroughputTransload(
                                totalSortie3Time, planPay, augmented, staging);
}
else {
    cargoThruTransSrt1 = 0.0;
    cargoThruTransSrt2 = 0.0;
    cargoThruTransSrt3 = 0.0;
}
srt1 = new sortie(fromIdent, selectedAirfields[m][0].id,
                  selectedAirfields[m][0].distFromPrim, cargoThruTransSrt1,
                  totalSortie1Time, totalSortie1Fuel, sortie1Param);
srt2 = new sortie(selectedAirfields[m][0].id, selectedAirfields[m][1][n].id,
                  selectedAirfields[m][1][n].distPrimSec, cargoThruTransSrt2,
                  totalSortie2Time, totalSortie2Fuel, sortie2Param);
srt3 = new sortie(selectedAirfields[m][1][n].id, toIdent,
                  selectedAirfields[m][1][n].distSecTo, cargoThruTransSrt3,
                  totalSortie3Time, totalSortie3Fuel, sortie3Param);

// add sortie to Array
srtArr = [];
srtArr.push(srt1);
srtArr.push(srt2);
srtArr.push(srt3);

// create route and add sortie to route
totalDistance = srt1.distance + srt2.distance + srt3.distance;
maxPayloadNoTrans = Math.min(sortie1Param.payload, sortie2Param.payload,
                             sortie3Param.payload);
routeFuelNoTrans = srt1.enrouteFuel + srt2.enrouteFuel + srt3.enrouteFuel;
var rte = new route(srtArr, 2, totalDistance, 0, 0, maxPayloadNoTrans, 0, 0, 0,
                    0);

// determine delivery and cycle times for the route
rte.deliveryTime = selectMDS.RouteDeliveryTimeCalculator(srtArr, augmented,
                                                         transload, staging);
rte.cycleTime = selectMDS.RouteCycleTimeCalculator(srtArr, augmented, transload,
                                                   staging);
```

```
                    // determine cargo throughput
                    maxTransCargoThru = Math.max(srt1.cargoThroughput, srt2.cargoThroughput,
                                                 srt3.cargoThroughput);
                    maxTransCargoThruPlanPay = Math.max(cargoThruTransPlanPaySrt1,
                                                cargoThruTransPlanPaySrt2, cargoThruTransPlanPaySrt3);
                    if (transload) {
                        rte.cargoThroughput = maxTransCargoThru /
                                              ((maxTransCargoThru / srt1.cargoThroughput) +
                                               (maxTransCargoThru / srt2.cargoThroughput) +
                                               (maxTransCargoThru / srt3.cargoThroughput));
                        rte.cargoThruPlanPay = maxTransCargoThruPlanPay /
                                              ((maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt1) +
                                               (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt2) +
                                               (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt3));
                    }
                    else {
                        rte.cargoThroughput = 24 * rte.maxPayloadNoTrans / rte.cycleTime;
                        if (planPay > rte.maxPayloadNoTrans)
                            rte.cargoThruPlanPay = rte.cargoThroughput;
                        else
                            rte.cargoThruPlanPay = 24 * planPay / rte.cycleTime;
                    }

                    // determine fuel efficiency
                    routeFuelTrans = selectMDS.DailyFuelConsumptionTrans(srtArr, augmented, staging,
                                                                         maxTransCargoThru);
                    routeFuelTransPlanPay = selectMDS.DailyFuelConsumptionTrans(srtArr, augmented,
                                                                staging, maxTransCargoThruPlanPay);
                    if (transload) {
                        rte.fuelEfficiency = rte.cargoThroughput / routeFuelTrans;
                        rte.fuelEffPlanPay = rte.cargoThruPlanPay / routeFuelTransPlanPay;
                    }
                    else {
                        rte.fuelEfficiency = rte.cargoThroughput /
                                             (24 * 2 * routeFuelNoTrans / rte.cycleTime);
                        rte.fuelEffPlanPay = rte.cargoThruPlanPay /
                                             (24 * 2 * routeFuelNoTrans / rte.cycleTime);
                    }

                    // add route to Array of routes
                    routes.push(rte);
                }
            }
        }
    }
}
else if (numStopForRoute == 3) {
    // determine if a secondary airfield exists and the first sortie is possible zero payload
    if (selectedAirfields[p][1].length != 0 && (selectedAirfields[p][0].distFromPrim <=  maxRange)) {
        // loop through the secondary airfields
        for (var q = 0; q < selectedAirfields[p][1].length; q++) {
            // determine if a tertiary airfield exists and the second sortie is possible zero payload
            if (selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][1].length != 0 &&
                (selectedAirfields[p][1][q].distPrimSec <= maxRange)) {
                // loop through the tertiary airfields
              for (var r = 0; r < selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][1].length;
                   r++) {
                    // is the third sortie possible zero payload
                    if (selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][1][r].distPrimSec <=
                        maxRange) {
                        // is the fourth sortie possible zero payload
                        if (selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][1][r].distSecTo
                            <= maxRange) {
                            // determine sortie parameters
                            sortie1Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
                                           maxPCNtakeoffWeight,
                                           selectedAirfields[p][0].altFromPrim,
                                           selectedAirfields[p][0].distFromPrim,
```

```
                    airfields[fromIdent].elev / 1000,
                    airfields[selectedAirfields[p][0].id].elev / 1000,
                    climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
sortie2Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
                    selectedAirfields[p][0].maxPCNgw,
                    selectedAirfields[p][1][q].altPrimSec,
                    selectedAirfields[p][1][q].distPrimSec,
                    airfields[selectedAirfields[p][0].id].elev / 1000,
                    airfields[selectedAirfields[p][1][q].id].elev / 1000,
                    climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
sortie3Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
         selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][0].
                         maxPCNgw
         selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][1][r].
                         altPrimSec,
         selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][1][r].
                         distPrimSec,
         airfields[selectedAirfields[p][1][q].id].elev / 1000,
         airfields[selectedAirfields[selectedAirfields[p][1][q].
                         selAfldsPos][1][r].id].elev / 1000,
         climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);

sortie4Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
      selectedAirfields[selectedAirfields[selectedAirfields[p][1][q].
                        selAfldsPos][1][r].selAfldsPos][0].maxPCNgw,
      selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][1][r].
                        altSecTo,
      selectedAirfields[selectedAirfields[p][1][q].selAfldsPos][1][r].
                        distSecTo,
      airfields[selectedAirfields[selectedAirfields[p][1][q].
                          selAfldsPos][1][r].id].elev / 1000,
      airfields[toIdent].elev / 1000,
      climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);

// create sortie
totalSortie1Time = sortie1Param.TotalTime();
totalSortie1Fuel = sortie1Param.TotalFuel();
totalSortie2Time = sortie2Param.TotalTime();
totalSortie2Fuel = sortie2Param.TotalFuel();
totalSortie3Time = sortie3Param.TotalTime();
totalSortie3Fuel = sortie3Param.TotalFuel();
totalSortie4Time = sortie4Param.TotalTime();
totalSortie4Fuel = sortie4Param.TotalFuel();

if (transload) {
    cargoThruTransSrt1 = selectMDS.CargoThroughputTransload(
              totalSortie1Time, sortie1Param.payload, augmented, staging);
    cargoThruTransSrt2 = selectMDS.CargoThroughputTransload(
              totalSortie2Time, sortie2Param.payload, augmented, staging);
    cargoThruTransSrt3 = selectMDS.CargoThroughputTransload(
              totalSortie3Time, sortie3Param.payload, augmented, staging);
    cargoThruTransSrt4 = selectMDS.CargoThroughputTransload(
              totalSortie4Time, sortie4Param.payload, augmented, staging);
    if (planPay > sortie1Param.payload)
        cargoThruTransPlanPaySrt1 = cargoThruTransSrt1;
    else
        cargoThruTransPlanPaySrt1 = selectMDS.CargoThroughputTransload(
              totalSortie1Time, planPay, augmented, staging);
    if (planPay > sortie2Param.payload)
        cargoThruTransPlanPaySrt2 = cargoThruTransSrt2;
    else
        cargoThruTransPlanPaySrt2 = selectMDS.CargoThroughputTransload(
              totalSortie2Time, planPay, augmented, staging);
    if (planPay > sortie3Param.payload)
        cargoThruTransPlanPaySrt3 = cargoThruTransSrt3;
    else
        cargoThruTransPlanPaySrt3 = selectMDS.CargoThroughputTransload(
              totalSortie3Time, planPay, augmented, staging);
    if (planPay > sortie4Param.payload)
```

```
        cargoThruTransPlanPaySrt4 = cargoThruTransSrt4;
    else
        cargoThruTransPlanPaySrt4 = selectMDS.CargoThroughputTransload(
              totalSortie4Time, planPay, augmented, staging);
}
else {
    cargoThruTransSrt1 = 0.0;
    cargoThruTransSrt2 = 0.0;
    cargoThruTransSrt3 = 0.0;
    cargoThruTransSrt4 = 0.0;
}
srt1 = new sortie(fromIdent, selectedAirfields[p][0].id,
                selectedAirfields[p][0].distFromPrim, cargoThruTransSrt1,
                totalSortie1Time, totalSortie1Fuel, sortie1Param);
srt2 = new sortie(selectedAirfields[p][0].id,
               selectedAirfields[p][1][q].id,
               selectedAirfields[p][1][q].distPrimSec, cargoThruTransSrt2,
               totalSortie2Time, totalSortie2Fuel, sortie2Param);
srt3 = new sortie(selectedAirfields[p][1][q].id,
                  selectedAirfields[selectedAirfields[p][1][q].
                                    selAfldsPos][1][r].id,
                  selectedAirfields[selectedAirfields[p][1][q].
                                    selAfldsPos][1][r].distPrimSec,
                  cargoThruTransSrt3, totalSortie3Time,
                  totalSortie3Fuel, sortie3Param);
srt4 = new sortie(selectedAirfields[selectedAirfields[p][1][q].
                                    selAfldsPos][1][r].id,
                  toIdent,
                  selectedAirfields[selectedAirfields[p][1][q].
                                    selAfldsPos][1][r].distSecTo,
                  cargoThruTransSrt4, totalSortie4Time,
                  totalSortie4Fuel, sortie4Param);

// add sortie to Array
srtArr = [];
srtArr.push(srt1);
srtArr.push(srt2);
srtArr.push(srt3);
srtArr.push(srt4);

// create route and add sortie to route
totalDistance = srt1.distance + srt2.distance + srt3.distance +
                srt4.distance;

maxPayloadNoTrans = Math.min(sortie1Param.payload, sortie2Param.payload,
                             sortie3Param.payload, sortie4Param.payload);
routeFuelNoTrans = srt1.enrouteFuel + srt2.enrouteFuel +
                   srt3.enrouteFuel + srt4.enrouteFuel;
var rte = new route(srtArr, 3, totalDistance, 0, 0, maxPayloadNoTrans, 0,
                    0, 0, 0);
// determine delivery and cycle times for the route
rte.deliveryTime = selectMDS.RouteDeliveryTimeCalculator(srtArr,
                                          augmented, transload, staging);
rte.cycleTime = selectMDS.RouteCycleTimeCalculator(srtArr, augmented,
                                          transload, staging);
// determine cargo throughput
maxTransCargoThru = Math.max(srt1.cargoThroughput, srt2.cargoThroughput,
                             srt3.cargoThroughput, srt4.cargoThroughput);
maxTransCargoThruPlanPay = Math.max(cargoThruTransPlanPaySrt1,
                                    cargoThruTransPlanPaySrt2,
                                    cargoThruTransPlanPaySrt3,
                                    cargoThruTransPlanPaySrt4);
if (transload) {
    rte.cargoThroughput = maxTransCargoThru /
                          ((maxTransCargoThru / srt1.cargoThroughput) +
                           (maxTransCargoThru / srt2.cargoThroughput) +
                           (maxTransCargoThru / srt3.cargoThroughput) +
                           (maxTransCargoThru / srt4.cargoThroughput));
    rte.cargoThruPlanPay = maxTransCargoThruPlanPay /
```

```
                                        ((maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt1) +
                                         (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt2) +
                                         (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt3) +
                                         (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt4));
                                }
                                else {
                                    rte.cargoThroughput = 24 * rte.maxPayloadNoTrans / rte.cycleTime;
                                    if (planPay > rte.maxPayloadNoTrans)
                                        rte.cargoThruPlanPay = rte.cargoThroughput;
                                    else
                                        rte.cargoThruPlanPay = 24 * planPay / rte.cycleTime;
                                }
                                // determine fuel efficiency
                                routeFuelTrans = selectMDS.DailyFuelConsumptionTrans(srtArr, augmented,
                                                            staging, maxTransCargoThru);
                                routeFuelTransPlanPay = selectMDS.DailyFuelConsumptionTrans(srtArr,
                                                            augmented, staging, maxTransCargoThruPlanPay);
                                if (transload) {
                                    rte.fuelEfficiency = rte.cargoThroughput / routeFuelTrans;
                                    rte.fuelEffPlanPay = rte.cargoThruPlanPay / routeFuelTransPlanPay;
                                }
                                else {
                                    rte.fuelEfficiency = rte.cargoThroughput /
                                                         (24 * 2 * routeFuelNoTrans / rte.cycleTime);
                                    rte.fuelEffPlanPay = rte.cargoThruPlanPay /
                                                         (24 * 2 * routeFuelNoTrans / rte.cycleTime);
                                }
                                // add route to Array of routes
                                routes.push(rte);
                            }
                        }
                    }
                }
            }
        }
    }
}
else if (numStopForRoute == 4) {
    // determine if a secondary airfield exists and the first sortie is possible zero payload
    if (selectedAirfields[s][1].length != 0 && (selectedAirfields[s][0].distFromPrim <= maxRange)) {
        // loop through the secondary airfields
        for (var t = 0; t < selectedAirfields[s][1].length; t++) {
            // determine if a tertiary airfield exists and the second sortie is possible zero payload
            if (selectedAirfields[selectedAirfields[s][1][t].selAfldsPos][1].length != 0 &&
                                        (selectedAirfields[s][1][t].distPrimSec <= maxRange)) {
                // loop through the tertiary airfields
                for (var u = 0; u < selectedAirfields[selectedAirfields[s][1][t].selAfldsPos][1].
                                                                                    length; u++) {
                    // determine if quaternary airfield exists & third sortie is possible zero payload
                    if (selectedAirfields[selectedAirfields[selectedAirfields[s][1][t].
                                            selAfldsPos][1][u].selAfldsPos][1].length != 0 &&
                        (selectedAirfields[selectedAirfields[s][1][t].selAfldsPos][1][u].distPrimSec
                         <= maxRange)) {
                        // loop through the quaternary airfields
                        for (var v = 0; v < selectedAirfields[selectedAirfields[
                          selectedAirfields[s][1][t].selAfldsPos][1][u].selAfldsPos][1].length; v++) {
                            // is the fourth sortie possible zero payload
                            if (selectedAirfields[selectedAirfields[selectedAirfields[s][1][t].
                                selAfldsPos][1][u].selAfldsPos][1][v].distPrimSec <= maxRange) {
                                // is the fifth sortie possible zero payload
                                if (selectedAirfields[selectedAirfields[selectedAirfields[s][1][t].
                                    selAfldsPos][1][u].selAfldsPos][1][v].distSecTo <= maxRange) {
                                    // determine sortie parameters
                                    sortie1Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
                                                maxPCNtakeoffWeight,
                                                selectedAirfields[s][0].altFromPrim,
                                                selectedAirfields[s][0].distFromPrim,
                                                airfields[fromIdent].elev / 1000,
                                                airfields[selectedAirfields[s][0].id].elev / 1000,
                                                climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
```

```
sortie2Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
               selectedAirfields[s][0].maxPCNgw,
               selectedAirfields[s][1][t].altPrimSec,
               selectedAirfields[s][1][t].distPrimSec,
               airfields[selectedAirfields[s][0].id].elev / 1000,
               airfields[selectedAirfields[s][1][t].id].elev/1000,
               climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
sortie3Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
               selectedAirfields[selectedAirfields[s][1][t].
                                 selAfldsPos][0].maxPCNgw,
               selectedAirfields[selectedAirfields[s][1][t].
                                 selAfldsPos][1][u].altPrimSec,
               selectedAirfields[selectedAirfields[s][1][t].
                                 selAfldsPos][1][u].distPrimSec,
               airfields[selectedAirfields[s][1][t].id].elev/1000,
               airfields[selectedAirfields[
                                 selectedAirfields[s][1][t].
                                 selAfldsPos][1][u].id].elev/1000,
               climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
sortie4Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
               selectedAirfields[selectedAirfields[
                                 selectedAirfields[s][1][t].
                                 selAfldsPos][1][u].
                                 selAfldsPos][0].maxPCNgw,
               selectedAirfields[selectedAirfields[
                                 selectedAirfields[s][1][t].
                                 selAfldsPos][1][u].
                                 selAfldsPos][1][v].altPrimSec,
               selectedAirfields[selectedAirfields[
                                 selectedAirfields[s][1][t].
                                 selAfldsPos][1][u].
                                 selAfldsPos][1][v].distPrimSec,
               airfields[selectedAirfields[
                                 selectedAirfields[s][1][t].
                                 selAfldsPos][1][u].id].elev/1000,
               airfields[selectedAirfields[
                                 selectedAirfields[
                                 selectedAirfields[s][1][t].
                                 selAfldsPos][1][u].
                                 selAfldsPos][1][v].id].elev/1000,
                climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);
sortie5Param = selectMDS.FuelConsumedIterationGrossWeightFixed(
               selectedAirfields[selectedAirfields[
                     selectedAirfields[selectedAirfields[s][1][t].
                     selAfldsPos][1][u].selAfldsPos][1][v].
                     selAfldsPos][0].maxPCNgw,
               selectedAirfields[selectedAirfields[
                     selectedAirfields[s][1][t].
                     selAfldsPos][1][u].
                     selAfldsPos][1][v].altSecTo,
               selectedAirfields[selectedAirfields[
                     selectedAirfields[s][1][t].
                     selAfldsPos][1][u].
                     selAfldsPos][1][v].distSecTo,
               airfields[selectedAirfields[selectedAirfields[
                     selectedAirfields[s][1][t].
                     selAfldsPos][1][u].
                     selAfldsPos][1][v].id].elev / 1000,
               airfields[toIdent].elev / 1000,
               climbAWF, enrouteAWF, descendAWF, enrouteDeltaT);

// create sortie
totalSortie1Time = sortie1Param.TotalTime();
totalSortie1Fuel = sortie1Param.TotalFuel();
totalSortie2Time = sortie2Param.TotalTime();
totalSortie2Fuel = sortie2Param.TotalFuel();
totalSortie3Time = sortie3Param.TotalTime();
totalSortie3Fuel = sortie3Param.TotalFuel();
totalSortie4Time = sortie4Param.TotalTime();
```

```
totalSortie4Fuel = sortie4Param.TotalFuel();
totalSortie5Time = sortie5Param.TotalTime();
totalSortie5Fuel = sortie5Param.TotalFuel();

if (transload) {
    cargoThruTransSrt1 = selectMDS.CargoThroughputTransload(
      totalSortie1Time, sortie1Param.payload, augmented, staging);
    cargoThruTransSrt2 = selectMDS.CargoThroughputTransload(
      totalSortie2Time, sortie2Param.payload, augmented, staging);
    cargoThruTransSrt3 = selectMDS.CargoThroughputTransload(
      totalSortie3Time, sortie3Param.payload, augmented, staging);
    cargoThruTransSrt4 = selectMDS.CargoThroughputTransload(
      totalSortie4Time, sortie4Param.payload, augmented, staging);
    cargoThruTransSrt5 = selectMDS.CargoThroughputTransload(
      totalSortie5Time, sortie5Param.payload, augmented, staging);
    if (planPay > sortie1Param.payload)
        cargoThruTransPlanPaySrt1 = cargoThruTransSrt1;
    else
        cargoThruTransPlanPaySrt1 = selectMDS.
               CargoThroughputTransload(totalSortie1Time, planPay,
                                        augmented, staging);
    if (planPay > sortie2Param.payload)
        cargoThruTransPlanPaySrt2 = cargoThruTransSrt2;
    else
        cargoThruTransPlanPaySrt2 = selectMDS.
               CargoThroughputTransload(totalSortie2Time, planPay,
                                        augmented, staging);
    if (planPay > sortie3Param.payload)
        cargoThruTransPlanPaySrt3 = cargoThruTransSrt3;
    else
        cargoThruTransPlanPaySrt3 = selectMDS.
               CargoThroughputTransload(totalSortie3Time, planPay,
                                        augmented, staging);
    if (planPay > sortie4Param.payload)
        cargoThruTransPlanPaySrt4 = cargoThruTransSrt4;
    else
        cargoThruTransPlanPaySrt4 = selectMDS.
               CargoThroughputTransload(totalSortie4Time, planPay,
                                        augmented, staging);
    if (planPay > sortie5Param.payload)
        cargoThruTransPlanPaySrt5 = cargoThruTransSrt5;
    else
        cargoThruTransPlanPaySrt5 = selectMDS.
               CargoThroughputTransload(totalSortie5Time, planPay,
                                        augmented, staging);
}
else {
    cargoThruTransSrt1 = 0.0;
    cargoThruTransSrt2 = 0.0;
    cargoThruTransSrt3 = 0.0;
    cargoThruTransSrt4 = 0.0;
    cargoThruTransSrt5 = 0.0;
}
srt1 = new sortie(fromIdent, selectedAirfields[s][0].id,
                  selectedAirfields[s][0].distFromPrim,
                  cargoThruTransSrt1,
                  totalSortie1Time, totalSortie1Fuel,
                  sortie1Param);
srt2 = new sortie(selectedAirfields[s][0].id,
                  selectedAirfields[s][1][t].id,
                  selectedAirfields[s][1][t].distPrimSec,
                  cargoThruTransSrt2,
                  totalSortie2Time, totalSortie2Fuel,
                  sortie2Param);
srt3 = new sortie(selectedAirfields[s][1][t].id,
                  selectedAirfields[selectedAirfields[s][1][t].
                                selAfldsPos][1][u].id,
                  selectedAirfields[selectedAirfields[s][1][t].
                                selAfldsPos][1][u].distPrimSec,
```

```
                    cargoThruTransSrt3,
                    totalSortie3Time, totalSortie3Fuel,
                    sortie3Param);
srt4 = new sortie(selectedAirfields[selectedAirfields[s][1][t].
                                selAfldsPos][1][u].id,
                    selectedAirfields[selectedAirfields[
                                selectedAirfields[s][1][t].
                                selAfldsPos][1][u].
                                selAfldsPos][1][v].id,
                    selectedAirfields[selectedAirfields[
                                selectedAirfields[s][1][t].
                                selAfldsPos][1][u].
                                selAfldsPos][1][v].distSecTo,
                    cargoThruTransSrt4,
                    totalSortie4Time, totalSortie4Fuel,
                    sortie4Param);
srt5 = new sortie(selectedAirfields[selectedAirfields[
                                selectedAirfields[s][1][t].
                                selAfldsPos][1][u].
                                selAfldsPos][1][v].id,
                    toIdent,
                    selectedAirfields[selectedAirfields[
                                selectedAirfields[s][1][t].
                                selAfldsPos][1][u].
                                selAfldsPos][1][v].distSecTo,
                    cargoThruTransSrt5,
                    totalSortie5Time, totalSortie5Fuel,
                    sortie5Param);

// add sortie to Array
srtArr = [];
srtArr.push(srt1);
srtArr.push(srt2);
srtArr.push(srt3);
srtArr.push(srt4);
srtArr.push(srt5);

// create route and add sortie to route
totalDistance = srt1.distance + srt2.distance + srt3.distance +
                srt4.distance + srt5.distance;
maxPayloadNoTrans = Math.min(sortie1Param.payload,
                     sortie2Param.payload, sortie3Param.payload,
                     sortie4Param.payload, sortie5Param.payload);
routeFuelNoTrans = srt1.enrouteFuel + srt2.enrouteFuel +
                   srt3.enrouteFuel + srt4.enrouteFuel +
                   srt5.enrouteFuel;
var rte = new route(srtArr, 4, totalDistance, 0, 0,
                    maxPayloadNoTrans, 0, 0, 0, 0);

// determine delivery and cycle times for the route
rte.deliveryTime = selectMDS.RouteDeliveryTimeCalculator(srtArr,
                                    augmented, transload, staging);
rte.cycleTime = selectMDS.RouteCycleTimeCalculator(srtArr,
                                    augmented, transload, staging);

// determine cargo throughput
maxTransCargoThru = Math.max(srt1.cargoThroughput,
                     srt2.cargoThroughput, srt3.cargoThroughput,
                     srt4.cargoThroughput, srt5.cargoThroughput);
maxTransCargoThruPlanPay = Math.max(cargoThruTransPlanPaySrt1,
            cargoThruTransPlanPaySrt2, cargoThruTransPlanPaySrt3,
            cargoThruTransPlanPaySrt4, cargoThruTransPlanPaySrt5);
if (transload) {
    rte.cargoThroughput = maxTransCargoThru /
                     ((maxTransCargoThru / srt1.cargoThroughput) +
                      (maxTransCargoThru / srt2.cargoThroughput) +
                      (maxTransCargoThru / srt3.cargoThroughput) +
                      (maxTransCargoThru / srt4.cargoThroughput) +
                      (maxTransCargoThru / srt5.cargoThroughput));
```

```
                                    rte.cargoThruPlanPay = maxTransCargoThruPlanPay /
                                         ((maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt1) +
                                          (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt2) +
                                          (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt3) +
                                          (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt4) +
                                          (maxTransCargoThruPlanPay / cargoThruTransPlanPaySrt5));
                                }
                                else {
                                    rte.cargoThroughput = 24*rte.maxPayloadNoTrans/rte.cycleTime;
                                    if (planPay > rte.maxPayloadNoTrans)
                                        rte.cargoThruPlanPay = rte.cargoThroughput;
                                    else
                                        rte.cargoThruPlanPay = 24 * planPay / rte.cycleTime;
                                }

                                // determine fuel efficiency
                                routeFuelTrans = selectMDS.DailyFuelConsumptionTrans(srtArr,
                                            augmented, staging, maxTransCargoThru);
                                routeFuelTransPlanPay = selectMDS.DailyFuelConsumptionTrans(
                                            srtArr, augmented, staging, maxTransCargoThruPlanPay);
                                if (transload) {
                                    rte.fuelEfficiency = rte.cargoThroughput / routeFuelTrans;
                                    rte.fuelEffPlanPay = rte.cargoThruPlanPay /
                                                         routeFuelTransPlanPay;
                                }
                                else {
                                    rte.fuelEfficiency = rte.cargoThroughput /
                                                        (24 * 2 * routeFuelNoTrans / rte.cycleTime);
                                    rte.fuelEffPlanPay = rte.cargoThruPlanPay /
                                                        (24 * 2 * routeFuelNoTrans / rte.cycleTime);
                                }

                                // add route to Array of routes
                                routes.push(rte);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Appendix B: Aircraft Performance Algorithms

AircraftMDS Object

Property	Description
operatingWeight	The empty weight of the aircraft withouth cargo or fuel.
maxFuelLoad	The maximum fuel load that an aircraft can carry.
acftMaxPayload	The maximum payload that an aircraft can carry.
maxGrossTakeoffWeight	The maximum gross weight that an aircraft can have at takeoff.
fuelSTTO	Fuel for start, taxi and takeoff.
fuelRAH	Fuel for reserve, alternate and holding.
rangeBeta0 - 5	Specific range regression Betas.
rangeWindAdjustBeta0 and 1	Specific range wind adjustment regression Betas.
rangeTempAdjustBeta0 and 1	Specific range temperature adjustment regression Betas.
machBeta0 - 2	Mach airspeed regression Betas to achieve specific range.
trueAirspeedBeta0 - 3	True airspeed regression Betas for given mach.
timeToClimbBeta0 - 3	Time to climb to cruise altitude regression Betas.
distToClimbBeta0 - 3	Distance to climb to cruise altitude regression Betas.
fuelToClimbBeta0 - 3	Fuel to climb to cruise altitude regression Betas.
timeToDescendBeta0 - 4	Time to descend from cruise altitude regression Betas.
distToDescendBeta0 - 4	Distance to climb to cruise altitude regression Betas.
fuelToDescendBeta0 - 4	Fuel to climb to cruise altitude regression Betas.
timeApproach	Time for an approach to a landing.
fuelApproach	Fuel for an approach to a landing.
optimumAltitudeBeta0 and 1	Optimum altitude regression Betas given gross takeoff weight.
CFLBeta0 - 9	Critical field length regression Betas.
landDistBeta0 - 5	Landing distance regression Betas
minRwyLength	Minimum runway length for takeoff and landing.
minRwyWidth	Minimum runway width for takeoff and landing.
gndTimeRefuel	Ground time for fueling service only.
gndTimeTransload	Ground time for fueling and cargo loading/unloading.
normFDP	Normal crew flight duty period.
augFDP	Augmented crew flight duty period.
showToTakeoffTime	Time from aircrew show at the airfield to takeoff.
crewRestToTakeoffTime	Time from aircrew entering crew rest to subsequent takeoff.

```
// Creates an aircraftMDS() object with the properties described
function aircraftMDS(operatingWeight, maxFuelLoad, acftMaxPayload, maxGrossTakeoffWeight,fuelSTTO,fuelRAH,
      rangeBeta0, rangeBeta1, rangeBeta2, rangeBeta3, rangeBeta4, rangeBeta5,
      rangeWindAdjustBeta0, rangeWindAdjustBeta1, rangeTempAdjustBeta0, rangeTempAdjustBeta1,
      machBeta0, machBeta1, machBeta2,
      trueAirspeedBeta0, trueAirspeedBeta1, trueAirspeedBeta2, trueAirspeedBeta3,
      timeToClimbBeta0, timeToClimbBeta1, timeToClimbBeta2, timeToClimbBeta3,
      distToClimbBeta0, distToClimbBeta1, distToClimbBeta2, distToClimbBeta3,
      fuelToClimbBeta0, fuelToClimbBeta1, fuelToClimbBeta2, fuelToClimbBeta3,
      timeToDescendBeta0, timeToDescendBeta1, timeToDescendBeta2, timeToDescendBeta3, timeToDescendBeta4,
      distToDescendBeta0, distToDescendBeta1, distToDescendBeta2, distToDescendBeta3, distToDescendBeta4,
```

```
      fuelToDescendBeta0, fuelToDescendBeta1, fuelToDescendBeta2, fuelToDescendBeta3, fuelToDescendBeta4,
      timeApproach, fuelApproach, optimumAltitudeBeta0, optimumAltitudeBeta1,
      CFLBeta0, CFLBeta1, CFLBeta2, CFLBeta3, CFLBeta4, CFLBeta5, CFLBeta6, CFLBeta7, CFLBeta8, CFLBeta9,
      landDistBeta0, landDistBeta1, landDistBeta2, landDistBeta3, landDistBeta4, landDistBeta5,
      minRwyLength, minRwyWidth, gndTimeRefuel, gndTimeTransload, normFDP, augFDP, showToTakeOffTime,
      crewRestToTakeoffTime) {
    this.operatingWeight = operatingWeight;
    this.maxFuelLoad = maxFuelLoad;
    this.acftMaxPayload = acftMaxPayload;
    this.maxGrossTakeoffWeight = maxGrossTakeoffWeight;
    this.fuelSTTO = fuelSTTO;
    this.fuelRAH = fuelRAH;
    this.rangeBeta0 = rangeBeta0; this.rangeBeta1 = rangeBeta1; this.rangeBeta2 = rangeBeta2;
    this.rangeBeta3 = rangeBeta3; this.rangeBeta4 = rangeBeta4;
    this.rangeBeta5 = rangeBeta5;
    this.rangeWindAdjustBeta0 = rangeWindAdjustBeta0; this.rangeWindAdjustBeta1 = rangeWindAdjustBeta1;
    this.rangeTempAdjustBeta0 = rangeTempAdjustBeta0; this.rangeTempAdjustBeta1 = rangeTempAdjustBeta1;
    this.machBeta0 = machBeta0; this.machBeta1 = machBeta1; this.machBeta2 = machBeta2;
    this.trueAirspeedBeta0 = trueAirspeedBeta0; this.trueAirspeedBeta1 = trueAirspeedBeta1;
    this.trueAirspeedBeta2 = trueAirspeedBeta2;
    this.trueAirspeedBeta3 = trueAirspeedBeta3;
    this.timeToClimbBeta0 = timeToClimbBeta0; this.timeToClimbBeta1 = timeToClimbBeta1;
    this.timeToClimbBeta2 = timeToClimbBeta2;
    this.timeToClimbBeta3 = timeToClimbBeta3;
    this.distToClimbBeta0 = distToClimbBeta0; this.distToClimbBeta1 = distToClimbBeta1;
    this.distToClimbBeta2 = distToClimbBeta2;
    this.distToClimbBeta3 = distToClimbBeta3;
    this.fuelToClimbBeta0 = fuelToClimbBeta0; this.fuelToClimbBeta1 = fuelToClimbBeta1;
    this.fuelToClimbBeta2 = fuelToClimbBeta2;
    this.fuelToClimbBeta3 = fuelToClimbBeta3;
    this.timeToDescendBeta0 = timeToDescendBeta0; this.timeToDescendBeta1 = timeToDescendBeta1;
    this.timeToDescendBeta2 = timeToDescendBeta2;
    this.timeToDescendBeta3 = timeToDescendBeta3; this.timeToDescendBeta4 = timeToDescendBeta4;
    this.distToDescendBeta0 = distToDescendBeta0; this.distToDescendBeta1 = distToDescendBeta1;
    this.distToDescendBeta2 = distToDescendBeta2;
    this.distToDescendBeta3 = distToDescendBeta3; this.distToDescendBeta4 = distToDescendBeta4;
    this.fuelToDescendBeta0 = fuelToDescendBeta0; this.fuelToDescendBeta1 = fuelToDescendBeta1;
    this.fuelToDescendBeta2 = fuelToDescendBeta2;
    this.fuelToDescendBeta3 = fuelToDescendBeta3; this.fuelToDescendBeta4 = fuelToDescendBeta4;
    this.timeApproach = timeApproach;
    this.fuelApproach = fuelApproach;
    this.optimumAltitudeBeta0 = optimumAltitudeBeta0; this.optimumAltitudeBeta1 = optimumAltitudeBeta1;
    this.CFLBeta0 = CFLBeta0; this.CFLBeta1 = CFLBeta1; this.CFLBeta2 = CFLBeta2;
    this.CFLBeta3 = CFLBeta3; this.CFLBeta4 = CFLBeta4;
    this.CFLBeta5 = CFLBeta5; this.CFLBeta6 = CFLBeta6; this.CFLBeta7 = CFLBeta7;
    this.CFLBeta8 = CFLBeta8; this.CFLBeta9 = CFLBeta9;
    this.landDistBeta0 = landDistBeta0; this.landDistBeta1 = landDistBeta1;
    this.landDistBeta2 = landDistBeta2; this.landDistBeta3 = landDistBeta3;
    this.landDistBeta4 = landDistBeta4; this.landDistBeta5 = landDistBeta5;
    this.minRwyLength = minRwyLength;
    this.minRwyWidth = minRwyWidth;
    this.gndTimeRefuel = gndTimeRefuel;
    this.gndTimeTransload = gndTimeTransload;
    this.normFDP = normFDP;
    this.augFDP = augFDP;
    this.showToTakeoffTime = showToTakeOffTime;
    this.crewRestToTakeoffTime = crewRestToTakeoffTime;
}
```

AircraftMDS Instances

```
/*Creates a C-5 object*/
var C5 = new aircraftMDS(380.0, 347.0, 270.0, 769.0, 3.0, 64.49,
   24.537604095439, 0.551086914994, 0.000196228333, -0.031831015851, 0.000019136753, -0.000490331600,
   1.0, 0.002286, 1.0, 0.0,
   0.168379101527, 0.010106030835, 0.000400547000,
   0.739962476547, -0.130206378986851, 661.129455909944, -2.28799249530959,
   1.252387836193, 0.181229214659, 0.000040478237, 0.000001163904,
   3.298199257838, 1.112839809943, 0.000090469247, 0.000012012901,
   1.061809140053, 0.113892311887, 0.000051725044, -0.000000368473,
   -4.11367589780242, 0.0185940177135711, -0.000018340182240486, 0.228172699660533, 0.00055088879246535,
   -19.8953296408788, 0.0766858214859992, -0.0000726639271038045, 1.37705092013579, 0.00257255876872813,
   -1.96730051813474, 0.0127768282753516, -0.0000134326424870467, 0.125354145077721, 0.000408173205033307,
   15.0, 7.0, 61, -0.0425,
   -5103.5465, -103.2944, -11.783695, -15.5467, 16.658, 0.341386389, 0.03894492, 0.051381575, 0.0, 0.0,
   -260.714285714287, 6.25, 0.0, -49.25, 0.803571428571439, 0.21875,
   6000, 147, 3.25, 4.25, 16, 24, 4.25, 17);

/*Creates a C-17 object*/
var C17 = new aircraftMDS(282.5, 241.36, 170.9, 585.0, 4.5, 58.96,
   31.73467029167, 0.989743669608, -0.0043149137723, -0.0642240850275, 0.00005804928, -0.00111004228507,
   1.0, 0.00225, 1.0, -0.001,
   0.246320503490156, 0.00906275433728003, 0.000458483810445982,
   0.739962476547397, -0.130206378986851, 661.129455909944, -2.28799249530959,
   0.768165733568, 0.187137734086, 0.000126560805, 0.000000358138,
   0.381646463716, 1.250278887724, 0.000552747494, 0.000011378658,
   0.646288319323, 0.093066858168, 0.000103839433, -0.000001181866,
   0.730068577, 0.014312226, -0.000021362504580405, 0.204191375, 0.000592283,
   -16.38205887, 0.127751984, -0.000174384, 1.39198126, 0.003565526,
   0.2573857831, 0.000459419581, -0.000000850006, 0.01076770546, 0.000032978865,
   10.0, 2.67, 57.5, -0.048333,
   -3453.1687, -207.14686, -13.52897, -19.67703, 15.21445, 0.849110, 0.055456217, 0.080657577, 0.0, 0.0,
   2647.910009, -4.1201061901, 0.0095053095053, -72.7475403384485, 2.01200314836676, 0.300681818181817,
   3500, 90, 2.25, 3.25, 16, 24, 2.75, 16.5);

/*Creates a C-130 object*/
var C130 = new aircraftMDS(78.0, 43.0, 53.0, 164.0, 0.8, 8.0,
   58.828846543414, 3.529168288956, -0.009825092212, -0.238433503513, 0.000973972948, -0.015457782260,
   1.0, 0.003, 1.0, -0.001,
   0.220902738806, 0.005456388441, 0.001010573113,
   0.739962476547397, -0.130206378986851, 661.129455909944, -2.28799249530959,
   0.919939347620, 0.177576636227, 0.000347380647, 0.000237542230,
   4.068901921019, 0.286553529526, -0.002770909951, 0.001170985342,
   0.173207750106, 0.016002475941, 0.000702896272, 0.000001175950,
   -2.983877886, -0.045011956, 0.000509625, 1.447135018, -0.00558139,
   -22.07517477, -0.081347346, 0.001624119, 4.462689967, -0.014344866,
   -0.051271152, -0.001162746, 0.0000138251262642665, 0.03669346629413, -0.000155475,
   10.0, 0.7, 55.5, -0.175,
   2113.034985, -620.5379, 10.93586, -64.2196285, 6.3673469, 5.8367347, 0.0, 0.604045, 0.1171256, 2.26351,
   2023.656462585, 0.31150793651, 0.0615079365079, -93.200255102041, 2.28635204081633, 1.13571428571429,
   3000, 80, 1.5, 2.25, 16, 18, 2.25, 16);
```

AircraftMDS Functions

Name	Inputs	Description
CriticalFieldLength	(gross weight, elevation, temperature)	Returns the critical field length.
LandingDistance	(gross weight, elevation)	Returns landing distance over a 50 ft obstacle.
TimeToClimb	(gross weight, altitude, elevation)	Returns time to climb to a given altitude.
DistToClimb	(gross weight, altitude, elevation, climb average wind factor, time to climb)	Returns distance to climb to a given altitude
FuelToClimb	(gross weight, altitude, elevation)	Returns fuel to climb to a given altitude
TimeToDescend	(gross weight, altitude, elevation)	Returns time to descend from a given altitude.
DistToDescend	(gross weight, altitude, elevation, descent average wind factor, time to descend)	Returns distance to descend from a given altitude
FuelToDescend	(gross weight, altitude, elevation)	Returns fuel to descend from a given altitude.
MachAirspeed	(gross weight, altitude, fuel consumed)	Returns Mach airspeed for 99% max range.
TrueAirspeed	(altitude, mach)	Returns true airspeed for a given Mach.
TaxiSpeed	()	Returns taxi speed.
ClimbSpeed	()	Returns climb speed.
DescentSpeed	(gross weight)	Returns descent speed.
ApproachSpeed	(gross weight)	Returns approach speed.
OptimumAltitude	(distance, gross weight, azimuth)	Returns optimum alt based off weight & direction.
OptimumAltitudeMaxGW	(distance, azimuth)	Returns optimum alt based off max weight & direction.
MaxDistance	(altitude, reserve alternate holding fuel, payload, fuelConsumed, average wind factor, delta t)	Returns the maximum distance that can be flown at a given altitude given a payload and fuel consumed.
FuelConsumed	(altitude, reserve alternate holding fuel, payload, distance, average wind factor, delta t)	Returns the fuel consumed at a given altitude, given payload, winds, temp and distance.
MaxDistanceForSortie	(maxPCNweight, elev1, elev2, climbAWF, enrouteAWF, descendAWF, enrouteDeltaT, alt)	Returns the maximum distance that can be flown at zero payload considering pavement strength, climb & descent.
AircraftCapableFor SortieZeroPayload	(maxPCNweight, dist, elev1, elev2, climbAWF, enrouteAWF, descendAWF, enrouteDeltaT, az)	Returns boolean true if aircraft is capable of sortie zero payload using the optimum altitude.
AircraftCapableFor SortieZeroPayload AltKnown	(maxPCNweight, dist, elev1, elev2, climbAWF, enrouteAWF, descendAWF, enrouteDeltaT, alt)	Returns boolean true if aircraft is capable of sortie zero payload using the given altitude.
AircraftCapableFor SortieGivenPayload	(maxPCNweight, dist, payload, elev1, elev2, climbAWF, enrouteAWF, descendAWF, enrouteDeltaT, az)	Returns boolean true if aircraft is capable of sortie given payload using the optimum altitude.
AircraftCanClimb AndDescend GivenDistance AndAltitude	(dist, alt, payload, elev1, elev2, climbAWF, enrouteAWF, descendAWF, enrouteDeltaT)	Returns boolean true if the distance to climb to and descend from a given altitude is less than the enroute distance
RouteCycle TimeCalculator	(srtArr, augmentedBool, transloadBool, stageBool)	Returns the cycle time from origin to destination & back.
RouteDelivery TimeCalculator	(srtArr, augmentedBool, transloadBool, stageBool)	Returns the delivery time from origin to destination.

Name	Inputs	Description
CargoThroughputTransload	(enrouteTime, payload, augmentedBool, stagingBool)	Returns cargo throughput with transload.
GWwithinRange	(gw)	Returns boolean if gross weight is between Op wt and max.
PayloadWithinRange	(payload)	Returns boolean if payload is between zero and acft max.
DailyFuelConsumptionTrans	(srtArr, augmented, stagebool)	Returns daily fuel consumption with transload.
FuelConsumedIteration GrossWeightFixed	(maxPCNWeight, alt, distance, elev1, elev2, climbAWF, enrouteAWF, descendAWF, deltat)	Returns sortie parameters object that includes all information for the sortie given an initial fixed gross weight.
FuelConsumedIteration PayloadFixed	(payload, alt, distance, elev1, elev2, climbAWF, enrouteAWF, descendAWF, deltat)	Returns sortie parameters object that includes all information for the sortie given an initial fixed payload.
RunwayLengthIsShorter ThanRegMin	(rwyLength)	Returns boolean true if runway length is longer than regulation minimum.
RunwayWidthIsShorter ThanRegMin	(rwyWidth)	Returns boolean true if runway width is wider than regulation minimum.
AirfieldPavementCan SupportTakeoff	(maxPCNweight)	Returns boolean true if airfield pavement strength can support takeoff

```
// Determines the critical field length given the gross weight (gw) in Klbs, elevation of airfield (elev)
// in 1,000s of feet & temperature of the airfield (temp) in Deg C
aircraftMDS.prototype.CriticalFieldLength = function (gw, elev, temp) {
    var criticalFieldLength = 0.0;
    criticalFieldLength = this.CFLBeta0 + this.CFLBeta1 * elev + this.CFLBeta2 * Math.pow(elev, 2) +
                          this.CFLBeta3 * temp + this.CFLBeta4 * gw + this.CFLBeta5 * gw * elev +
                          this.CFLBeta6 * gw * Math.pow(elev, 2) + this.CFLBeta7 * gw * temp +
                          this.CFLBeta8 * Math.pow(temp, 2) + this.CFLBeta9 * temp * elev;

    return criticalFieldLength;
}


// Determines the landing distance in feet given the gross weight (gw) in Klbs & elevation of airfield
// (elev) in 1,000s of feet.
aircraftMDS.prototype.LandingDistance = function (gw, elev) {
    var landingDistance = 0.0;
    landingDistance = this.landDistBeta0 + this.landDistBeta1 * gw +
                      this.landDistBeta2 * Math.pow(gw, 2) + this.landDistBeta3 * elev +
                      this.landDistBeta4 * Math.pow(elev, 2) + this.landDistBeta5 * gw * elev;

    return landingDistance;
}

// Determines the time to climb given the gross weight (gw) in Klbs, enroute altitude (alt) in 1,000s of
// feet & the elevation of airfield (elev) in 1,000s of feet.
aircraftMDS.prototype.TimeToClimb = function (gw, alt, elev) {
    var timeToClimb = 0.0;
    if (alt > 5.0) {
        timeToClimb = this.timeToClimbBeta0 + this.timeToClimbBeta1 * alt +
                      this.timeToClimbBeta2 * Math.pow(alt, 2) * Math.pow(gw, 3) / 1000000 +
                      this.timeToClimbBeta3 * Math.pow(alt, 3) * Math.pow(gw, 3) / 1000000;
        if (elev >= 1.0) {
            timeToClimb = timeToClimb - (this.timeToClimbBeta0 + this.timeToClimbBeta1 * elev +
                          this.timeToClimbBeta2 * Math.pow(elev, 2) * Math.pow(gw, 3) / 1000000 +
                          this.timeToClimbBeta3 * Math.pow(elev, 3) * Math.pow(gw, 3) / 1000000);
        }
```

```
    }
    else {
        timeToClimb = ((alt - elev) / 5) * (this.timeToClimbBeta0 + this.timeToClimbBeta1 * 5 +
                      this.timeToClimbBeta2 * 25 * Math.pow(gw, 3) / 1000000 +
                      this.timeToClimbBeta3 * 125 * Math.pow(gw, 3) / 1000000);
    }

    if ((alt - elev) < 0)
        timeToClimb = 0;

    return timeToClimb;
}

// Determines the distance to climb given the gross weight (gw) in Klbs, enroute altitude (alt) in 1,000s
// of feet, the elevation of airfield (elev) in 1,000s of feet, the climb average wind factor and the time
// to climb in minutes.
aircraftMDS.prototype.DistToClimb = function (gw, alt, elev, climbAWF, TimeToClimb) {
    var distToClimb = 0.0;
    if (alt > 5.0) {
        distToClimb = this.distToClimbBeta0 + this.distToClimbBeta1 * alt +
                      this.distToClimbBeta2 * Math.pow(alt, 2) * Math.pow(gw, 3) / 1000000 +
                      this.distToClimbBeta3 * Math.pow(alt, 3) * Math.pow(gw, 3) / 1000000 +
                      (climbAWF * TimeToClimb / 60);
        if (elev >= 1.0) {
            distToClimb = distToClimb - (this.distToClimbBeta0 + this.distToClimbBeta1 * elev +
                          this.distToClimbBeta2 * Math.pow(elev, 2) * Math.pow(gw, 3) / 1000000 +
                          this.distToClimbBeta3 * Math.pow(elev, 3) * Math.pow(gw, 3) / 1000000 +
                          (climbAWF * TimeToClimb / 60));
        }
    }
    else {
        distToClimb = ((alt - elev) / 5) * (this.distToClimbBeta0 + this.distToClimbBeta1 * 5 +
                      this.distToClimbBeta2 * 25 * Math.pow(gw, 3) / 1000000 +
                      this.distToClimbBeta3 * 125 * Math.pow(gw, 3) / 1000000) +
                      (climbAWF * TimeToClimb / 60);
    }

    if ((alt - elev) < 0)
        distToClimb = 0;

    return distToClimb;
}



// Determines the fuel to climb given the gross weight (gw) in Klbs, enroute altitude (alt) in 1,000s of
// feet & the elevation of airfield (elev) in 1,000s of feet.
aircraftMDS.prototype.FuelToClimb = function (gw, alt, elev) {
    var fuelToClimb = 0.0;
    if (alt > 5.0) {
        fuelToClimb = this.fuelToClimbBeta0 + this.fuelToClimbBeta1 * alt +
                      this.fuelToClimbBeta2 * Math.pow(alt, 2) * Math.pow(gw, 3) / 1000000 +
                      this.fuelToClimbBeta3 * Math.pow(alt, 3) * Math.pow(gw, 3) / 1000000;
        if (elev >= 1.0) {
            fuelToClimb = fuelToClimb - (this.fuelToClimbBeta0 + this.fuelToClimbBeta1 * elev +
                          this.fuelToClimbBeta2 * Math.pow(elev, 2) * Math.pow(gw, 3) / 1000000 +
                          this.fuelToClimbBeta3 * Math.pow(elev, 3) * Math.pow(gw, 3) / 1000000);
        }
    }
    else {
        fuelToClimb = ((alt - elev) / 5) * (this.fuelToClimbBeta0 + this.fuelToClimbBeta1 * 5 +
                      this.fuelToClimbBeta2 * 25 * Math.pow(gw, 3) / 1000000 +
                      this.fuelToClimbBeta3 * 125 * Math.pow(gw, 3) / 1000000);
    }

    if ((alt - elev) < 0)
        fuelToClimb = 0;

    return fuelToClimb;
```

```
}

// Determines the time to descend given the gross weight (gw) in Klbs, enroute altitude (alt) in 1,000s of
// feet & the elevation of airfield (elev) in 1,000s of feet.
aircraftMDS.prototype.TimeToDescend = function (gw, alt, elev) {
    var timeToDescend = 0.0;
    if (alt > 5.0) {
        timeToDescend = this.timeToDescendBeta0 + this.timeToDescendBeta1 * gw +
                        this.timeToDescendBeta2 * Math.pow(gw, 2) +
                        this.timeToDescendBeta3 * alt + this.timeToDescendBeta4 * gw * alt;
        if (elev >= 1.0) {
            timeToDescend = timeToDescend - (this.timeToDescendBeta0 + this.timeToDescendBeta1 * gw +
                            this.timeToDescendBeta2 * Math.pow(gw, 2) +
                            this.timeToDescendBeta3 * elev + this.timeToDescendBeta4 * gw * elev);
        }
    }
    else {
        timeToDescend = ((alt - elev) / 5) * (this.timeToDescendBeta0 + this.timeToDescendBeta1 * gw +
                        this.timeToDescendBeta2 * Math.pow(gw, 2) +
                        this.timeToDescendBeta3 * 5 + this.timeToDescendBeta4 * gw * 5);
    }

    if ((alt - elev) < 0)
        timeToDescend = 0;

    return timeToDescend;
}

// Determines the distance to descend given the gross weight (gw) in Klbs, enroute altitude (alt) in
// 1,000s of feet, the elevation of airfield (elev) in 1,000s of feet, the average wind factor for descent
// and the time to descend in minutes.
aircraftMDS.prototype.DistToDescend = function (gw, alt, elev, descendAWF, timeToDescend) {
    var distToDescend = 0.0;
    if (alt > 5.0) {
        distToDescend = this.distToDescendBeta0 + this.distToDescendBeta1 * gw +
                        this.distToDescendBeta2 * Math.pow(gw, 2) + this.distToDescendBeta3 * alt +
                        this.distToDescendBeta4 * gw * alt + (descendAWF * timeToDescend / 60);
        if (elev >= 1.0) {
            distToDescend = distToDescend - (this.distToDescendBeta0 + this.distToDescendBeta1 * gw +
                            this.distToDescendBeta2 * Math.pow(gw, 2) +
                            this.distToDescendBeta3 * elev + this.distToDescendBeta4 * gw * elev +
                            (descendAWF * timeToDescend / 60));
        }
    }
    else {
        distToDescend = ((alt - elev) / 5) * (this.distToDescendBeta0 + this.distToDescendBeta1 * gw +
                        this.distToDescendBeta2 * Math.pow(gw, 2) +
                        this.distToDescendBeta3 * 5 + this.distToDescendBeta4 * gw * 5) +
                        (descendAWF * timeToDescend / 60);
    }

    if ((alt - elev) < 0)
        distToDescend = 0;
    return distToDescend;
}

// Determine the fuel to descend given the gross weight (gw) in Klbs, enroute altitude (alt) in 1,000s of
// feet & the elevation of airfield (elev) in 1,000s of feet.
aircraftMDS.prototype.FuelToDescend = function (gw, alt, elev) {
    var fuelToDescend = 0.0;
    if (alt > 5.0) {
        fuelToDescend = this.fuelToDescendBeta0 + this.fuelToDescendBeta1 * gw +
                        this.fuelToDescendBeta2 * Math.pow(gw, 2) + this.fuelToDescendBeta3 * alt +
                        this.fuelToDescendBeta4 * gw * alt;
        if (elev >= 1.0) {
            fuelToDescend = fuelToDescend - (this.fuelToDescendBeta0 + this.fuelToDescendBeta1 * gw +
                            this.fuelToDescendBeta2 * Math.pow(gw, 2) + this.fuelToDescendBeta3 * elev +
                            this.fuelToDescendBeta4 * gw * elev);
        }
```

```
    }
    else {
        fuelToDescend = ((alt - elev) / 5) * (this.fuelToDescendBeta0 + this.fuelToDescendBeta1 * gw +
                        this.fuelToDescendBeta2 * Math.pow(gw, 2) + this.fuelToDescendBeta3 * 5 +
                        this.fuelToDescendBeta4 * gw * 5);
    }

    if ((alt - elev) < 0)
        fuelToDescend = 0;

    return fuelToDescend;
}

// Determine the mach airspeed given the gross weight (gw) in Klbs, enroute altitude (alt) in 1,000s of
// feet & the fuel consumed (fc) in 1,000s of pounds.
aircraftMDS.prototype.MachAirspeed = function (gw, alt, fc) {
    var mach = 0.0;
    mach = this.machBeta0 + this.machBeta1 * alt + this.machBeta2 * (gw - fc / 2.0);
    return mach;
}

// Determine the true airspeed given the gross weight (gw) in Klbs, enroute altitude (alt) in 1,000s of
// feet & the fuel consumed (fc) in 1,000s of pounds.
aircraftMDS.prototype.TrueAirspeed = function (alt, mach) {
    var tas = 0.0;
    tas = this.trueAirspeedBeta0 + this.trueAirspeedBeta1 * alt + this.trueAirspeedBeta2 * mach +
          this.trueAirspeedBeta3 * alt * mach;
    return tas;
}

// Given the MDS, returns a string of the taxi speed
aircraftMDS.prototype.TaxiSpeed = function () {
    var taxiSpeed = "";
    switch (this) {
        case C5:
            taxiSpeed = "Max 30 Knots"; break;
        case C17:
            taxiSpeed = "Max 40 Knots"; break;
        case C130:
            taxiSpeed = "Max 20 Knots:<br/>20 deg nose deflection<br />Max 5 Knots:<br/>60 deg nose
                         deflection"; break;
        default: break;
    }
    return taxiSpeed;
}

// Given the MDS, returns a string of the climb speed
aircraftMDS.prototype.ClimbSpeed = function () {
    var climbSpeed = "";
    switch (this) {
        case C5:
            climbSpeed = "250 KCAS to 10,<br />then 270 to 29,<br />then .7 Mach"; break;
        case C17:
            climbSpeed = "250 KCAS to 10,<br />then .74 Mach"; break;
        case C130:
            climbSpeed = "180 KIAS to 10,<br />then 170 KIAS to 15<br />then 160 KIAS to 25<br />4 eng
                          climb above 25"; break;
        default: break;
    }
    return climbSpeed;
}

// Given the MDS and gross weight at start of descent, returns a string of the descent speed
aircraftMDS.prototype.DescentSpeed = function (gw) {
    var descentSpeed = "";
    var c130jMaxRangeDescent = 91.217 + 0.565217 * gw;
    switch (this) {
        case C5:
            descentSpeed = "0.72 Mach or 300 KCAS<br />whichever is less"; break;
```

```
        case C17:
            descentSpeed = "0.74 Mach until<br />340 KCAS or 10,<br />250 KCAS below 10"; break;
        case C130:
            descentSpeed = c130jMaxRangeDescent.toFixed(0) + " KIAS"; break;
        default: break;
    }
    return descentSpeed;
}

// Given the MDS and gross weight in Klbs at start of approach, returns the approach speed
aircraftMDS.prototype.ApproachSpeed = function (gw) {
    var approachSpeed = 0.0;
    switch (this) {
        case C5:
            approachSpeed = 63.9 + 0.11111 * gw; break;
        case C17:
            approachSpeed = 75 + 0.126364 * gw; break;
        case C130:
            approachSpeed = 70 + 0.5 * gw; break;
        default: break;
    }
    return approachSpeed;
}

// Given the distance in NMs, the gross weight in KLbs and the azimuth, returns the optimum enroute
// altitude in 1,000s of feet
aircraftMDS.prototype.OptimumAltitude = function (dist, gw, az) {
    var optimumAltitude = 0.0;
    var optimumAltitudeByDist = dist / 10.0;
    optimumAltitude = this.optimumAltitudeBeta0 + this.optimumAltitudeBeta1 * gw;

    if (optimumAltitudeByDist < optimumAltitude)
        optimumAltitude = optimumAltitudeByDist;

    if ((az >= 0 && az < 180) || az == 360) {
        optimumAltitude = optimumAltitude - ((optimumAltitude + 1.0) % 2) + 2;
    }
    else {
        optimumAltitude = optimumAltitude - (optimumAltitude % 2) + 2;
    }
    return optimumAltitude;
}



// Given the distance in NMs and the azimuth, returns the optimum enroute altitude in 1,000s of feet for
// aircraft max gross weight
aircraftMDS.prototype.OptimumAltitudeMaxGW = function (dist, az) {
    var optimumAltitude = 0.0;
    var optimumAltitudeByDist = dist / 10.0;
    optimumAltitude = this.optimumAltitudeBeta0 + this.optimumAltitudeBeta1 * this.maxGrossTakeoffWeight;

    if (optimumAltitudeByDist < optimumAltitude)
        optimumAltitude = optimumAltitudeByDist;

    if ((az >= 0 && az < 180) || az == 360) {
        optimumAltitude = optimumAltitude - ((optimumAltitude + 1.0) % 2) + 2;
    }
    else {
        optimumAltitude = optimumAltitude - (optimumAltitude % 2) + 2;
    }
    return optimumAltitude;
}

// Given the altitude in 1,000s of feet, the fuel for reserve, alternate and holding (rah) in Klbs, the
// payload in Klbs, the fuel consumed in Klbs, the average wind factor (awf) and the delta t from standard
// day in deg C, determines the max distance possible
aircraftMDS.prototype.MaxDistance = function (altitude, rah, payload, fuelConsumed, awf, deltat) {
    var distance = 0.0;
```

```
    var tempAdjust = this.rangeTempAdjustBeta0 + deltat * this.rangeTempAdjustBeta1;
    var windAdjust = this.rangeWindAdjustBeta0 + awf * this.rangeWindAdjustBeta1;

    //Use Formula distance = A*g^3 + B*g^2 + C*g coefficients.  g represents fuel consumed.
    var A = tempAdjust * windAdjust * this.rangeBeta4 / 3.0;
    var B = tempAdjust * windAdjust * (this.rangeBeta3 / 2.0 + this.rangeBeta4 * (this.operatingWeight +
            payload + this.fuelSTTO + this.fuelApproach + rah) +
            this.rangeBeta5 * altitude / 2.0); //(β_3/2+β_4*(EW+w)+β_5/2*Alt),
    var C = tempAdjust * windAdjust * (this.rangeBeta0 + this.rangeBeta1 * altitude +
            this.rangeBeta2 * Math.pow(altitude, 2.0) +
            this.rangeBeta3 * (this.operatingWeight + payload + this.fuelSTTO + this.fuelApproach + rah) +
            this.rangeBeta4 * Math.pow(this.operatingWeight + payload + this.fuelSTTO +
            this.fuelApproach + rah, 2.0) +
            this.rangeBeta5 * altitude * (this.operatingWeight + payload + this.fuelSTTO +
            this.fuelApproach + rah));
    //(β_0+β_1*Alt+β_2*A〖lt〗^2+β_3*(EW+w)+β_4*(EW+w)^2+β_5*Alt*(EW+w))
    var G = fuelConsumed;

    distance = A * Math.pow(G, 3.0) + B * Math.pow(G, 2.0) + C * G;

    return distance;
}

// Given the altitude in 1,000s of feet, the fuel for reserve, alternate and holding (rah) in Klbs, the
// payload in Klbs, the distance in NMs, the average wind factor (awf) and the delta t from standard day
// in deg C, determines the fuel consumed in Klbs
aircraftMDS.prototype.FuelConsumed = function (altitude, rah, payload, distance, awf, deltat) {
    var fuelConsumed = 0.0;
    var tempAdjust = this.rangeTempAdjustBeta0 + deltat * this.rangeTempAdjustBeta1;
    var windAdjust = this.rangeWindAdjustBeta0 + awf * this.rangeWindAdjustBeta1;

    //Determine Cubic Formula A*g^3 + B*g^2 + C*g + D = 0 coefficients.  g represents fuel consumed.
    var A = tempAdjust * windAdjust * this.rangeBeta4 / 3.0;
    var B = tempAdjust * windAdjust * (this.rangeBeta3 / 2.0 + this.rangeBeta4 * (this.operatingWeight +
            payload + this.fuelSTTO + this.fuelApproach + rah) +
            this.rangeBeta5 * altitude / 2.0); //(β_3/2+β_4*(EW+w)+β_5/2*Alt),
    var C = tempAdjust * windAdjust * (this.rangeBeta0 + this.rangeBeta1 * altitude +
            this.rangeBeta2 * Math.pow(altitude, 2.0) +
            this.rangeBeta3 * (this.operatingWeight + payload + this.fuelSTTO + this.fuelApproach + rah) +
            this.rangeBeta4 * Math.pow(this.operatingWeight + payload + this.fuelSTTO +
            this.fuelApproach + rah, 2.0) +
            this.rangeBeta5 * altitude * (this.operatingWeight + payload + this.fuelSTTO +
            this.fuelApproach + rah));
    //(β_0+β_1*Alt+β_2*A〖lt〗^2+β_3*(EW+w)+β_4*(EW+w)^2+β_5*Alt*(EW+w))
    var D = -distance;

    var commonTerm1 = 2.0 * Math.pow(B, 3.0) - 9.0 * A * B * C + 27.0 * Math.pow(A, 2.0) * D;
    var commonTerm2 = 4.0 * Math.pow((Math.pow(B, 2.0) - 3.0 * A * C), 3.0);
    var cubeRoot1 = 0.5 * (commonTerm1 + Math.sqrt(Math.pow(commonTerm1, 2.0) - commonTerm2));
    var cubeRoot2 = 0.5 * (commonTerm1 - Math.sqrt(Math.pow(commonTerm1, 2.0) - commonTerm2));

    if (cubeRoot1 < 0)
        cubeRoot1 = -Math.pow(-cubeRoot1, (1.0 / 3.0));
    else
        cubeRoot1 = Math.pow(cubeRoot1, (1.0 / 3.0));

    if (cubeRoot2 < 0)
        cubeRoot2 = -Math.pow(-cubeRoot2, (1.0 / 3.0));
    else
        cubeRoot2 = Math.pow(cubeRoot2, (1.0 / 3.0));

    //g=-B/3A
    //  -1/3A ∛(1/2 [2B^3-9ABC+27A^2 D+√((2B^3-9ABC+27A^2 D)^2-4〖(B^2-3AC)〗^3 )] )
    //  -1/3A ∛(1/2 [2B^3-9ABC+27A^2 D-√((2B^3-9ABC+27A^2 D)^2-4〖(B^2-3AC)〗^3 )] )
    fuelConsumed = -(B / (3.0 * A)) - (1.0 / (3.0 * A)) * cubeRoot1 - (1.0 / (3.0 * A)) * cubeRoot2;

    return fuelConsumed;
}
```

```
// Determines the maximum distance that a given aircraft MDS can fly no payload at a given altitude
aircraftMDS.prototype.MaxDistanceForSortie = function (maxPCNweight, payload, elev1, elev2, climbAWF,
                                                       enrouteAWF, descendAWF, enrouteDeltaT, alt) {
    //Determine maximum gross takeoff weight
    var maxGrossTakeoffWeight = this.maxGrossTakeoffWeight;
    if (maxPCNweight < this.maxGrossTakeoffWeight)
        maxGrossTakeoffWeight = maxPCNweight;

    // Determine gross weight for maximum distance
    var gwForMaxDist = this.operatingWeight + this.maxFuelLoad + payload;
    if (gwForMaxDist > maxGrossTakeoffWeight)
        gwForMaxDist = maxGrossTakeoffWeight;

    // Determine Distance and Fuel for climb for maximum distance
    var maxDistTimeToClimb = this.TimeToClimb(gwForMaxDist, alt, elev1);
    var maxDistDistToClimb = this.DistToClimb(gwForMaxDist, alt, elev1, climbAWF, maxDistTimeToClimb);
    var maxDistFuelToClimb = this.FuelToClimb(gwForMaxDist, alt, elev1);

    // Determine Distance and Fuel for climb for maximum distance
    var gwForMaxDistDescent = gwForMaxDist - this.maxFuelLoad + this.fuelRAH;
    var maxDistTimeToDescend = this.TimeToDescend(gwForMaxDistDescent, alt, elev2);
    var maxDistDistToDescend = this.DistToDescend(gwForMaxDistDescent, alt, elev2, descendAWF,
                                                  maxDistTimeToDescend);
    var maxDistFuelToDescend = this.FuelToDescend(gwForMaxDistDescent, alt, elev2);

    //Determine the maximum distance filling the tanks to max for airfield
    var maxDistance = this.MaxDistance(alt, this.fuelRAH, payload,
                                       gwForMaxDist - this.operatingWeight - payload - this.fuelSTTO –
                                       maxDistFuelToClimb - maxDistFuelToDescend -
                                       this.fuelRAH, enrouteAWF, enrouteDeltaT) + maxDistDistToClimb +
                                       maxDistDistToDescend;

    return maxDistance;
}

// Determines if the aircraft MDS is capable of flying a given distance
aircraftMDS.prototype.AircraftCapableForSortieZeroPayload = function (maxPCNweight, dist, elev1, elev2,
                                                  climbAWF, enrouteAWF, descendAWF, enrouteDeltaT, az) {
    var aircraftCapable = true;

    //Determine maximum gross takeoff weight
    var maxGrossTakeoffWeight = this.maxGrossTakeoffWeight;
    if (maxPCNweight < this.maxGrossTakeoffWeight)
        maxGrossTakeoffWeight = maxPCNweight;

    // Determine gross weight for maximum distance
    var gwForMaxDist = this.operatingWeight + this.maxFuelLoad;
    if (gwForMaxDist > maxGrossTakeoffWeight)
        gwForMaxDist = maxGrossTakeoffWeight;

    // Determine Optimum Altitude for maximum distance zero payload
    var optimumAltForMaxDist = this.OptimumAltitude(dist, gwForMaxDist, az);

    //Determine the maximum distance using zero payload and filling the tanks to max for airfield
    var maxDistance = this.MaxDistanceForSortie(maxPCNweight, 0, elev1, elev2, climbAWF, enrouteAWF,
descendAWF, enrouteDeltaT, optimumAltForMaxDist);

    if (dist > maxDistance)
        aircraftCapable = false;
    return aircraftCapable;
}

// Determines if an aircraft MDS is capable of flying a sortie with no payload at a given altitude
aircraftMDS.prototype.AircraftCapableForSortieZeroPayloadAltKnown = function (maxPCNweight, dist, elev1,
                                        elev2, climbAWF, enrouteAWF, descendAWF, enrouteDeltaT, alt) {
    var aircraftCapable = true;

    //Determine the maximum distance using zero payload and filling the tanks to max for airfield
```

```
    var maxDistance = this.MaxDistanceForSortie(maxPCNweight, 0, elev1, elev2, climbAWF, enrouteAWF,
                                                descendAWF, enrouteDeltaT, alt);

    if (dist > maxDistance)
        aircraftCapable = false;
    return aircraftCapable;
}

// Determines if an aircraft MDS is capable of flying a sortie with a given payload at a given altitude
aircraftMDS.prototype.AircraftCapableForSortieGivenPayload = function (maxPCNweight, dist, payload, elev1,
                                            elev2, climbAWF, enrouteAWF, descendAWF, enrouteDeltaT, az) {
    var aircraftCapable = true;

    //Determine maximum gross takeoff weight
    var maxGrossTakeoffWeight = this.maxGrossTakeoffWeight;
    if (maxPCNweight < this.maxGrossTakeoffWeight)
        maxGrossTakeoffWeight = maxPCNweight;

    // Determine gross weight for maximum distance
    var gwForMaxDist = this.operatingWeight + this.maxFuelLoad + payload;
    if (gwForMaxDist > maxGrossTakeoffWeight)
        gwForMaxDist = maxGrossTakeoffWeight;

    // Determine Optimum Altitude for maximum distance zero payload
    var optimumAltForMaxDist = this.OptimumAltitude(dist, gwForMaxDist, az);

    //Determine the maximum distance using given payload and filling the tanks to max for airfield
    var maxDistance = this.MaxDistanceForSortie(maxPCNweight, payload, elev1, elev2, climbAWF, enrouteAWF,
                                                descendAWF, enrouteDeltaT, optimumAltForMaxDist);

    if (dist > maxDistance)
        aircraftCapable = false;
    return aircraftCapable;
}

// Determines if an aircraft MDS is capable of climbing to and descending from a given altitude over the
given distance
aircraftMDS.prototype.AircraftCanClimbAndDescendGivenDistanceAndAltitude = function (dist, alt, payload,
                                        elev1, elev2, climbAWF, enrouteAWF, descendAWF, enrouteDeltaT) {
    var aircraftCapable = true;

    var initFuelConsumed = this.FuelConsumed(alt, this.fuelRAH, payload, dist, enrouteAWF, enrouteDeltaT);
    var gw = this.operatingWeight + payload + this.fuelSTTO + this.fuelRAH + initFuelConsumed;
    var timeClimb = this.TimeToClimb(gw, alt, elev1);
    var distClimb = this.DistToClimb(gw, alt, elev1, climbAWF, timeClimb);
    var fuelClimb = this.FuelToClimb(gw, alt, elev1);
    var gwAtDescent = gw - this.fuelSTTO - fuelClimb - initFuelConsumed;
    var timeDescend = this.TimeToDescend(gwAtDescent, alt, elev2);
    var distDescend = this.DistToDescend(gwAtDescent, alt, elev2, descendAWF, timeDescend);
    var distEnroute = dist - distClimb - distDescend;

    if (distEnroute < 0.0)
        aircraftCapable = false;

    return aircraftCapable;
}

// Determines the cycle time for a given route
aircraftMDS.prototype.RouteCycleTimeCalculator = function (srtArr, augmentedBool, transloadBool,
                                                           stageBool) {
    var FDPtime;
    var cycleTime = 0.0;

    if (stageBool)
        cycleTime = RouteCycleTimeCalculatorStage(srtArr, this.showToTakeoffTime, this.gndTimeTransload,
                                                  this.gndTimeRefuel);
    else {
        if (augmentedBool)
            FDPtime = this.augFDP;
```

```
        else
            FDPtime = this.normFDP;
        cycleTime = RouteCycleTimeCalculatorNoStage(srtArr, this.showToTakeoffTime, FDPtime,
                                this.gndTimeTransload, this.gndTimeRefuel, this.crewRestToTakeoffTime);
    }

    return cycleTime;
}

// Determines the delivery time for a given route
aircraftMDS.prototype.RouteDeliveryTimeCalculator = function (srtArr, augmentedBool, transloadBool,
                                                               stageBool) {
    var FDPtime;
    var deliveryTime = 0.0;

    if (stageBool)
        deliveryTime = RouteDeliveryTimeCalculatorStage(srtArr, this.showToTakeoffTime,
                                                        this.gndTimeTransload, this.gndTimeRefuel);
    else {
        if (augmentedBool)
            FDPtime = this.augFDP;
        else
            FDPtime = this.normFDP;
        deliveryTime = RouteDeliveryTimeCalculatorNoStage(srtArr, this.showToTakeoffTime, FDPtime,
                            this.gndTimeTransload, this.gndTimeRefuel, this.crewRestToTakeoffTime);
    }
    return deliveryTime;
}

// Determines the cycle time for a given route assuming staging available at every stop
function RouteCycleTimeCalculatorStage(srtArr, showToTakeoff, gndTimeTrans, gndTimeRefuel) {
    var cycleTime = 0.0;

    for (i = 0; i < srtArr.length; i++) {
        cycleTime += srtArr[i].enrouteTime + gndTimeRefuel;
    }
    cycleTime = (cycleTime * 2) - (2 * gndTimeRefuel) + (2 * gndTimeTrans);

    return cycleTime;
}

// Determines the delivery time for a given route assuming staging available at every stop
function RouteDeliveryTimeCalculatorStage(srtArr, showToTakeoff, gndTimeTrans, gndTimeRefuel) {
    var deliveryTime = 0.0;

    for (i = 0; i < srtArr.length; i++) {
        deliveryTime += srtArr[i].enrouteTime + gndTimeRefuel;
    }
    deliveryTime += showToTakeoff - gndTimeRefuel;

    return deliveryTime;
}

// Determines the cycle time for a given route assuming staging is not available at every stop
function RouteCycleTimeCalculatorNoStage(srtArr, showToTakeoff, fdp, gndTimeTrans, gndTimeRefuel,
                                         crewRestToTakeoff) {
    var cycleTime = showToTakeoff;
    var fdpStartTime = 0.0;

    // Loop through sorties forward (source to destination)
    for (var i = 0; i < srtArr.length; i++) {
        cycleTime += srtArr[i].enrouteTime;
        if (i < (srtArr.length - 1)) {
            if (cycleTime - fdpStartTime + gndTimeRefuel + srtArr[i + 1].enrouteTime > fdp) {
                cycleTime += crewRestToTakeoff;
                fdpStartTime = cycleTime - showToTakeoff;
            }
            else
                cycleTime += gndTimeRefuel;
```

```
        }
        else {
            if (cycleTime - fdpStartTime + gndTimeTrans + srtArr[i].enrouteTime > fdp) {
                cycleTime += crewRestToTakeoff;
                fdpStartTime = cycleTime - showToTakeoff;
            }
            else
                cycleTime += gndTimeTrans;
        }
    }

    // Loop through sorties in reverse (destination to source)
    for (var j = srtArr.length - 1; j > -1; j--) {
        cycleTime += srtArr[j].enrouteTime;
        if (j > 0) {
            if (cycleTime - fdpStartTime + gndTimeRefuel + srtArr[j - 1].enrouteTime > fdp) {
                cycleTime += crewRestToTakeoff;
                fdpStartTime = cycleTime - showToTakeoff;
            }
            else {
                cycleTime += gndTimeRefuel;
            }
        }
        else {
            cycleTime += crewRestToTakeoff; ;
        }

    }
    cycleTime -= showToTakeoff;

    return cycleTime;
}

// Determines the delivery time for a given route assuming staging is not available at every stop
function RouteDeliveryTimeCalculatorNoStage(srtArr, showToTakeoff, fdp, gndTimeTrans, gndTimeRefuel,
                                            crewRestToTakeoff) {

    var deliveryTime = showToTakeoff;
    var fdpStartTime = 0.0;

    for (var i = 0; i < srtArr.length; i++) {
        deliveryTime += srtArr[i].enrouteTime;
        if (i < (srtArr.length - 1)) {
            if (deliveryTime - fdpStartTime + gndTimeRefuel + srtArr[i + 1].enrouteTime > fdp) {
                deliveryTime += crewRestToTakeoff;
                fdpStartTime = deliveryTime - showToTakeoff;
            }
            else
                deliveryTime += gndTimeRefuel;
        }
    }
    return deliveryTime;
}

// Determines the cargo throughput for a given route given transload operations
aircraftMDS.prototype.CargoThroughputTransload = function (enrouteTime, payload, augmentedBool,
                                                           stagingBool) {
    var cargoThru = 0.0;    var cycles = 0;          var sorties = 0;        var lastFlight = false;
    var FDPtime = 0.0;      var timeRemain = 0.0;    var cycleTime = 0.0;    var sortieCycleTime = 0.0;

    // determine time parameters
    crewRestToTakeoffTime = selectMDS.crewRestToTakeoffTime;
    if (augmentedBool)
        FDPtime = this.augFDP;
    else
        FDPtime = this.normFDP;

    // determine number of cycles
    timeRemain = FDPtime - this.showToTakeoffTime;
```

```
    cycleTime = enrouteTime + this.gndTimeTransload;
    cycles = Math.floor(timeRemain / cycleTime);

    //determine if the last flight is possible
    if (enrouteTime <= (timeRemain - (cycles * cycleTime)))
        lastFlight = true;
    else
        lastFlight = false;

    // determine the number of sorties
    if (lastFlight)
        sorties = cycles + 1;
    else
        sorties = cycles;

    // determine overall sortie cycle time
    if (stagingBool)
        sortieCycleTime = sorties * (enrouteTime + this.gndTimeTransload);
    else
        sortieCycleTime = sorties * enrouteTime + (sorties - 1) * this.gndTimeTransload +
                          this.crewRestToTakeoffTime;

    // calculate throughput
    cargoThru = 24 * payload * (sorties / 2) / sortieCycleTime;


    return cargoThru;
}

// Determine if the given gross weight is more than operating weight but less than aircraft max gross
// weight
aircraftMDS.prototype.GWwithinRange = function(gw) {
    if (gw >= this.operatingWeight && gw <= this.maxGrossTakeoffWeight)
        return true;
    else
        return false;
}

// Determines if the payload is more than 0 but less than the aircraft maximum allowed
aircraftMDS.prototype.PayloadWithinRange = function (payload) {
    if (payload >= 0 && payload <= this.acftMaxPayload)
        return true;
    else
        return false;
}

// Determines the daily fuel consumption given transload operations
aircraftMDS.prototype.DailyFuelConsumptionTrans = function (srtArr, augmented, stagebool, maxCargoThru) {
    var dailyFuelConsumption = 0.0;
    var FDPTime;
    var fuelSumArr = new Array();
    var fuelSum, currTime;
    var cargoThruAcft = new Array();
    var sumCargoThruAcft = 0;


    if (augmented) {
        FDPTime = this.augFDP;
    }
    else {
        FDPTime = this.normFDP;
    }
    fuelSum = 0.0;
    totalFuelSum = 0.0;

    for (i = 0; i < srtArr.length; i++) {
        cargoThruAcft[i] = maxCargoThru / srtArr[i].cargoThroughput;
        sumCargoThruAcft += cargoThruAcft[i];
        currTime = 0.0;
```

```
        fuelSum = 0.0;
        currTime += this.showToTakeoffTime;
        for (k = 0; k < 10; k++) {
            if (currTime + srtArr[i].enrouteTime < FDPTime) {
                currTime += srtArr[i].enrouteTime;
                fuelSum += srtArr[i].enrouteFuel;
                currTime += this.gndTimeTransload;
            }
        }
        if (stagebool) {
            currTime += -this.showToTakeoffTime
        }
        else {
            currTime += this.crewRestToTakeoffTime - this.gndTimeTransload - this.showToTakeoffTime;
        }
        fuelSumArr[i] = 24 * fuelSum / currTime;
    }
    for (i = 0; i < srtArr.length; i++) {
        totalFuelSum += fuelSumArr[i] * (cargoThruAcft[i] / sumCargoThruAcft);
    }
    dailyFuelConsumption = totalFuelSum;

    return dailyFuelConsumption;
}



// Determines sortie parameters by iterating through the fuel consumed function
aircraftMDS.prototype.FuelConsumedIterationGrossWeightFixed = function (maxPCNWeight, alt, distance,
                                            elev1, elev2, climbAWF, enrouteAWF, descendAWF, deltat) {
    var tempRampFuel = 0.0;
    var deltaRampFuel = 100.0;
    var maxRampFuelDelta = 0.1;

    //Loop to find actual fuel consumed
    var fuelSTTO = this.fuelSTTO;
    var fuelApproach = this.fuelApproach;
    var gw = this.maxGrossTakeoffWeight;

    if (maxPCNWeight < gw)
        gw = maxPCNWeight;
    var payload = gw - this.operatingWeight - fuelSTTO - fuelApproach - this.fuelRAH;
    if (payload > this.acftMaxPayload)
        payload = this.acftMaxPayload;
    var initFuelConsumed = this.FuelConsumed(alt, this.fuelRAH, payload, distance, enrouteAWF, deltat);

    var timeClimb = this.TimeToClimb(gw, alt, elev1);
    var distClimb = this.DistToClimb(gw, alt, elev1, climbAWF, timeClimb);
    var fuelClimb = this.FuelToClimb(gw, alt, elev1);
    var gwAtDescent = gw - fuelSTTO - fuelClimb - initFuelConsumed;
    var timeDescend = this.TimeToDescend(gwAtDescent, alt, elev2);
    var distDescend = this.DistToDescend(gwAtDescent, alt, elev2, descendAWF, timeDescend);
    var fuelDescend = this.FuelToDescend(gwAtDescent, alt, elev2);
    var distEnroute = distance - distClimb - distDescend;
    var fuelEnroute = this.FuelConsumed(alt, this.fuelRAH, payload, distEnroute, enrouteAWF, deltat);

    var rampFuel = fuelSTTO + fuelClimb + fuelEnroute + fuelDescend + fuelApproach + this.fuelRAH;
    payload = gw - this.operatingWeight - rampFuel;
    if (payload > this.acftMaxPayload)
        payload = this.acftMaxPayload;

    tempRampFuel = rampFuel;

    while (Math.abs(deltaRampFuel) > maxRampFuelDelta) {

        //Determine fuel required to fly given distance and payload
        fuelEnroute = this.FuelConsumed(alt, this.fuelRAH, payload, distEnroute, enrouteAWF, deltat)
        rampFuel = fuelSTTO + fuelClimb + fuelEnroute + fuelDescend + fuelApproach + this.fuelRAH;
```

```
        //Recalculate payload
        payload = gw - this.operatingWeight - rampFuel;
        if (payload >= this.acftMaxPayload) {
            payload = this.acftMaxPayload;
            //Recalculate gross weight
            gw = this.operatingWeight + rampFuel + payload;
        }

        //Recalculate climb parameteres given new gross weight
        timeClimb = this.TimeToClimb(gw, alt, elev1);
        distClimb = this.DistToClimb(gw, alt, elev1, climbAWF, timeClimb);
        fuelClimb = this.FuelToClimb(gw, alt, elev1);
        gwAtDescent = gw - fuelSTTO - fuelClimb - fuelEnroute;
        timeDescend = this.TimeToDescend(gwAtDescent, alt, elev2);
        distDescend = this.DistToDescend(gwAtDescent, alt, elev2, descendAWF, timeDescend);
        fuelDescend = this.FuelToDescend(gwAtDescent, alt, elev2);
        distEnroute = distance - distClimb - distDescend;

        deltaRampFuel = rampFuel - tempRampFuel;
        tempRampFuel = rampFuel;
    }
    var mach = this.MachAirspeed(gw, alt, fuelEnroute);
    var tas = this.TrueAirspeed(alt, mach);
    var groundSpeed = tas + enrouteAWF;
    var timeEnroute = distEnroute / groundSpeed;

    var sortieParam = new sortieParameters(this.operatingWeight, fuelSTTO, timeClimb, distClimb,
                                           fuelClimb, gw, payload, alt, rampFuel, timeEnroute,
                                           distEnroute, fuelEnroute, mach, tas, timeDescend, distDescend,
                                           fuelDescend, this.timeApproach, fuelApproach);

    return sortieParam;
}

// Determines sortie parameters by iterating through the fuel consumed function
aircraftMDS.prototype.FuelConsumedIterationPayloadFixed = function (payload, alt, distance, elev1, elev2,
                                                                    climbAWF, enrouteAWF, descendAWF, deltat) {
    var tempRampFuel = 0.0;
    var deltaRampFuel = 100.0;
    var maxRampFuelDelta = 0.1;

    //Loop to find actual fuel consumed
    var fuelSTTO = this.fuelSTTO;
    var fuelApproach = this.fuelApproach;
    var initFuelConsumed = this.FuelConsumed(alt, this.fuelRAH, payload, distance, enrouteAWF, deltat);

    var gw = this.operatingWeight + payload + fuelSTTO + initFuelConsumed + fuelApproach + this.fuelRAH;
    var timeClimb = this.TimeToClimb(gw, alt, elev1);
    var distClimb = this.DistToClimb(gw, alt, elev1, climbAWF, timeClimb);
    var fuelClimb = this.FuelToClimb(gw, alt, elev1);
    var gwAtDescent = gw - fuelSTTO - fuelClimb - initFuelConsumed;
    var timeDescend = this.TimeToDescend(gwAtDescent, alt, elev2);
    var distDescend = this.DistToDescend(gwAtDescent, alt, elev2, descendAWF, timeDescend);
    var fuelDescend = this.FuelToDescend(gwAtDescent, alt, elev2);
    var distEnroute = distance - distClimb - distDescend;
    var fuelEnroute = this.FuelConsumed(alt, this.fuelRAH, payload, distEnroute, enrouteAWF, deltat);
    var rampFuel = fuelSTTO + fuelClimb + fuelEnroute + fuelDescend + fuelApproach + this.fuelRAH;
    gw = this.operatingWeight + rampFuel + payload;

    tempRampFuel = rampFuel;

    while (Math.abs(deltaRampFuel) > maxRampFuelDelta) {
        //Determine fuel required to fly given distance and payload
        fuelEnroute = this.FuelConsumed(alt, this.fuelRAH, payload, distEnroute, enrouteAWF, deltat)
        rampFuel = fuelSTTO + fuelClimb + fuelEnroute + fuelDescend + fuelApproach + this.fuelRAH;

        //Recalculate gross weight
        gw = this.operatingWeight + rampFuel + payload;
```

```
        //Recalculate climb parameteres given new gross weight
        timeClimb = this.TimeToClimb(gw, alt, elev1);
        distClimb = this.DistToClimb(gw, alt, elev1, climbAWF, timeClimb);
        fuelClimb = this.FuelToClimb(gw, alt, elev1);
        gwAtDescent = gw - fuelSTTO - fuelClimb - fuelEnroute;
        timeDescend = this.TimeToDescend(gwAtDescent, alt, elev2);
        distDescend = this.DistToDescend(gwAtDescent, alt, elev2, descendAWF, timeDescend);
        fuelDescend = this.FuelToDescend(gwAtDescent, alt, elev2);
        distEnroute = distance - distClimb - distDescend;

        deltaRampFuel = rampFuel - tempRampFuel;
        tempRampFuel = rampFuel;
    }
    var mach = this.MachAirspeed(gw, alt, fuelEnroute);
    var tas = this.TrueAirspeed(alt, mach);
    var groundSpeed = tas + enrouteAWF;
    var timeEnroute = distEnroute / groundSpeed;

    var sortieParam = new sortieParameters(this.operatingWeight, fuelSTTO, timeClimb, distClimb,
                                           fuelClimb, gw, payload, alt, rampFuel, timeEnroute,
                                           distEnroute, fuelEnroute, mach, tas, timeDescend, distDescend,
                                           fuelDescend, this.timeApproach, fuelApproach);
    return sortieParam;
}


// determines if runway length is shorter than regulation minimums
aircraftMDS.prototype.RunwayLengthIsShorterThanRegMin = function (rwyLength) {
    if (rwyLength < this.minRwyLength)
        return true;
    else
        return false;
}


// determines if runway width is shorter than regulation minimums
aircraftMDS.prototype.RunwayWidthIsShorterThanRegMin = function (rwyWidth) {
    if (rwyWidth < this.minRwyWidth)
        return true;
    else
        return false;
}

// determines if airfield pavement can support takeoff
aircraftMDS.prototype.AirfieldPavementCanSupportTakeoff = function (maxPCNweight) {
    if (maxPCNweight > (this.operatingWeight + this.fuelRAH + this.fuelApproach + this.fuelSTTO))
        return true;
    else
        return false;
}
```

Sortie Parameters Object

Property	Description
opWeight	The empty weight of the aircraft without cargo or fuel.
fuelSTTO	Fuel for start, taxi and takeoff.
timeToClimb	Time to climb to cruise altitude for the sortie.
distToClimb	Distance to climb to cruise altitude for the sortie.
fuelToClimb	Fuel to climb to cruise altitude for the sortie.
gw	The aircraft gross weight at takeoff for the sortie.
payload	The payload for the sortie.
alt	The altitude for the sortie.
rampFuel	The ramp fuel for the sortie.
timeEnroute	Time of cruise at altitude for the sortie.
distEnroute	Distance of cruise at altitude for the sortie.
fuelEnroute	Fuel of cruise at altitude for the sortie.
machEnroute	Mach airspeed at cruise altitude for the sortie.
trueEnroute	True airspeed at cruise altitude for the sortie.
timeToDescend	Time to descend from cruise altitude for the sortie.
distToDescend	Distance to descend from cruise altitude for the sortie.
fuelToDescend	Fuel to descend from cruise altitude for the sortie.
timeApproach	Time for the approach and landing.
fuelApproach	Fuel for the approach and landing.

```
/* Creates sortie parameters object to hold all sortie parameters */
function sortieParameters(opWeight, fuelSTTO, timeToClimb, distToClimb, fuelToClimb, gw, payload, alt,
                          rampFuel, timeEnroute, distEnroute, fuelEnroute, machEnroute,
                 trueEnroute, timeToDescend, distToDescend, fuelToDescend, timeApproach, fuelApproach) {
    this.opWeight = opWeight;
    this.fuelSTTO = fuelSTTO;
    this.timeToClimb = timeToClimb; this.distToClimb = distToClimb; this.fuelToClimb = fuelToClimb;
    this.gw = gw;
    this.payload = payload;
    this.alt = alt;
    this.rampFuel = rampFuel;
    this.timeEnroute = timeEnroute; this.distEnroute = distEnroute; this.fuelEnroute = fuelEnroute;
    this.machEnroute = machEnroute; this.trueEnroute = trueEnroute;
    this.timeToDescend = timeToDescend; this.distToDescend = distToDescend;
    this.fuelToDescend = fuelToDescend;
    this.timeApproach = timeApproach;
    this.fuelApproach = fuelApproach;
}
```

Sortie Parameters Functions

Name	Inputs	Description
TotalDist	()	Returns total distance in NMs by summing climb, enroute & descent distances.
TotalTime	()	Returns total time in hours by summing climb, enroute and descent times.
TotalFuel	()	Returns total fuel in Klbs by summing climb, enroute and descent fuels.

```
// Determines sum of distances for climb, enroute and descent.
sortieParameters.prototype.TotalDist = function () {
    var totalDist = 0.0;
    totalDist = this.distToClimb + this.distEnroute + this.distToDescend;
    return totalDist;
}

// Determines sum of time for climb, enroute, descent and approach.
sortieParameters.prototype.TotalTime = function () {
    var totalTime = 0.0;
    totalTime = this.timeToClimb / 60 + this.timeEnroute + this.timeToDescend / 60 +
                this.timeApproach / 60;
    return totalTime;
}

// Determines sum of fuel for start, taxi, takeoff, climb, enroute, descent, approach and landing.
sortieParameters.prototype.TotalFuel = function () {
    var totalFuel = 0.0;
    totalFuel = this.fuelSTTO + this.fuelToClimb + this.fuelEnroute + this.fuelToDescend +
                this.fuelApproach;
    return totalFuel;
}
```

Appendix C: Airfield, Distance, Pavement and Airspace Algorithms

airfield Object

Property	Description
name	Name of the airfield.
state_prov	Number that represents the state where the airfield is located.
icao	Four letter ICAO designation.
wgs_dlat	Latitude in decimal degrees.
wgs_dlong	Longitude in decimal degrees.
elev	Elevation in feet.
type	Type of airfield (A - Civil, B - Civil and Military, C - Military, D - Inactive).
magvar	Magnetic variation of airfield location.
opr_agy	Operating agency.
runways	Array of runway objects.

```
// Airfield object
function airfield(name, state_prov, icao, wgs_dlat, wgs_dlong, elev, type, magvar, opr_agy, runways) {
    this.name = name;
    this.state_prov = state_prov;
    this.icao = icao;
    this.wgs_dlat = wgs_dlat;
    this.wgs_dlong = wgs_dlong;
    this.elev = elev;
    this.type = type;
    this.magvar = magvar;
    this.opr_agy = opr_agy;
    this.runways = runways;
}
```

runway Object

Property	Description
highIdent	The high end of the runway.
lowIdent	The low end of the runway
rwyLength	Runway length in feet.
width	Runway width in feet.
pcn	Runway Pavement Classification Number Code.

```
// Runway object
function runway(highIdent, lowIdent, rwyLength, width, pcn) {
    this.highIdent = highIdent;
    this.lowIdent = lowIdent;
    this.rwyLength = rwyLength;
    this.width = width;
    this.pcn = pcn;
}
```

GeodeticCurve Object

Property	Description
distance	The distance between lat-long pairs in meters.
azimuth	The azimuth from the origin to the destination.
reverseAzimuth	The reverse azimuth from the destination to the origin.
fromLat	The origin latitude in decimal degrees.
fromLong	The origin longitude in decimal degrees.
toLat	The destination latitude in decimal degrees.
toLong	The destination longitude in decimal degrees.
minLat	The minimum latitude on curve between origin and dest.
maxLat	The maximum latitude on curve between origin and dest.

```
// Creates geodetic curve object that represents a great circle segment
function GeodeticCurve(distance, azimuth, reverseAzimuth, fromLat, fromLong, toLat, toLong, minLat,
maxLat) {
    this.distance = distance;
    this.azimuth = azimuth;
    this.reverseAzimuth = reverseAzimuth;
    this.fromLat = fromLat;
    this.fromLong = fromLong;
    this.toLat = toLat;
    this.toLong = toLong;
    this.minLat = minLat;
    this.maxLat = maxLat;
}
```

waypoint Object

Property	Description
lat	The latitude in decimal degrees.
long	The longitude in decimal degrees.

```
// waypoint object
function waypoint(lat, long) {
    this.lat = lat;
    this.long = long;
}
```

longitudePair Object

Property	Description
long1	The first longitude in decimal degrees.
long2	The second longitude in decimal degrees.

```
// longitude pair object for calcLongGivenLat below
function LongitudePair(long1, long2) {
    this.long1 = long1;
    this.long2 = long2;
}
```

restrictedPolygon Object

Property	Description
waypoints	Array of waypoint objects representing polygon boundary.
geoCurves	Array of geodeticCurve objects representing polygon arcs.
maxLat	The maximum latitude of all geocurves.
minLat	The minimum latitude of all geocurves.
maxLong	The maximum longitude of all waypoints.
minLong	The minimum longitude of all waypoints.
potentialImpact	Boolean to identify potential intersection of sortie with restricted polygon.

```
// creates a restrictedPolygon object from an array of waypoint objects
function restrictedPolygon(waypoints) {
    if (waypoints) {
        this.waypoints = waypoints; //waypoints is an array of waypoint objects
        this.geoCurves = new Array();
        var geoCurve;
        this.maxLat = waypoints[0].lat;
        this.minLat = waypoints[0].lat;
        this.maxLong = waypoints[0].long;
        this.minLong = waypoints[0].long;
        if (waypoints.length > 1) {
            for (var i = 0; i < waypoints.length; i++) {
                if (i == waypoints.length - 1) {
                    geoCurve = new VincentyDistance(waypoints[i].lat, waypoints[i].long, waypoints[0].lat,
                                                    waypoints[0].long);
                    geoCurve.LatMaxMin();
                }
                else {
                    geoCurve = new VincentyDistance(waypoints[i].lat, waypoints[i].long, waypoints[i +
                                                    1].lat, waypoints[i + 1].long);
                    geoCurve.LatMaxMin();
                }
                this.geoCurves.push(geoCurve);
                if (geoCurve.maxLat > this.maxLat)
                    this.maxLat = geoCurve.maxLat;
                if (geoCurve.minLat < this.minLat)
                    this.minLat = geoCurve.minLat;
                if (waypoints[i].long > this.maxLong)
                    this.maxLong = waypoints[i].long;
                if (waypoints[i].long < this.minLong)
                    this.minLong = waypoints[i].long;
            }
        }
        this.potentialImpact = false;
    }
    else {
        this.waypoints = waypoints; //waypoints is an array of waypoint objects
        this.geoCurves = null;
        this.maxLat = null;
        this.minLat = null;
        this.maxLong = null;
        this.minLong = null;
        this.potentialImpact = false;
    }
}
```

flightSegment Object

Property	Description
fromLat	The origin latitude in decimal degrees on flight segment.
fromLong	The origin longitude in decimal degrees on flight segment.
toLat	The destination latitude in decimal degrees on flight segment.
toLong	The destination longitude in decimal degrees on flight segment.
dist	The distance in NMs of flight segment.

```
// flightSegment object representing portion of a sortie
function flightSegment(fromLat, fromLong, toLat, toLong, dist) {
    this.fromLat = fromLat;
    this.fromLong = fromLong;
    this.toLat = toLat;
    this.toLong = toLong;
    this.dist = dist;
}
```

flightPath Object

Property	Description
fltSegments	Array of flightSegment objects.
totalDist	Total distance of ll flight segments in NMs.

```
// flightPath object representing all fltSegments of a sortie
function flightPath(fltSegments, totalDist) {
    this.fltSegments = fltSegments;
    this.totalDist = totalDist;
}
```

Assorted Functions

Name	Inputs	Description
Vincenty Distance	(fromLat, fromLong, toLat, toLong)	Returns the Vincenty elliptical earth distance in meters between two lat-long pairs in decimal degrees.
DetermineLatLong Bounds	(fromLat, fromLong, toLat, toLong)	Returns google maps LatLngBounds object for setting map window size.
CalculateGreatCircle Midpoint	(fromLat, fromLong, toLat, toLong)	Returns text lat-long of midpoint for centering on Great Circle Mapper web site.
PCN to MaxGrossWeight	(selectMDS, PCN)	Returns aircraft maximum gross takeoff weight in Klbs given an aircraftMDS object and the airfield Pavement Classification Number (PCN)
Determine Average Temperature	(latitude, elevation)	Returns the average monthly temperature in degrees Celsius using the current month, a given latitude in decimal degrees and an elevation in feet.
HeadWind	(rwyHdg, windDirection, windSpeed)	Returns the headwind component in knots.
CrossWind	(rwyHdg, windDirection, windSpeed)	Returns the crosswind component in knots.
SelectShortestFlightPath	(fltPaths)	Returns the flightPath object with the minimum totalDist property from an Array of flightPath objects.

```
// Calculates the vincenty elliptical great circle geodetic curve (www.gavaghan.org)
function VincentyDistance(lat1, long1, lat2, long2) {

    //Ellipsoid properties based on WGS-84
    var semiMajor = 6378137.0; //Meters
    var inverseFlattening = 298.257223563;
    var flattening = 1.0 / inverseFlattening;
    var semiMinor = (1.0 - flattening) * semiMajor;

    // simplify
    var a = semiMajor;
    var b = semiMinor;
    var f = flattening;
    var TwoPi = 2.0 * Math.PI;

    // get parameters as radians
    var phi1 = lat1 * Math.PI / 180.0;
    var lambda1 = long1 * Math.PI / 180.0;
    var phi2 = lat2 * Math.PI / 180.0;
    var lambda2 = long2 * Math.PI / 180.0;

    // calculations
    var a2 = a * a;
    var b2 = b * b;
    var a2b2b2 = (a2 - b2) / b2;

    var omega = lambda2 - lambda1;
```

```
var tanphi1 = Math.tan(phi1);
var tanU1 = (1.0 - f) * tanphi1;
var U1 = Math.atan(tanU1);
var sinU1 = Math.sin(U1);
var cosU1 = Math.cos(U1);

var tanphi2 = Math.tan(phi2);
var tanU2 = (1.0 - f) * tanphi2;
var U2 = Math.atan(tanU2);
var sinU2 = Math.sin(U2);
var cosU2 = Math.cos(U2);

var sinU1sinU2 = sinU1 * sinU2;
var cosU1sinU2 = cosU1 * sinU2;
var sinU1cosU2 = sinU1 * cosU2;
var cosU1cosU2 = cosU1 * cosU2;

// eq. 13
var lambda = omega;

// intermediates we'll need to compute 's'
var A = 0.0;
var B = 0.0;
var sigma = 0.0;
var deltasigma = 0.0;
var lambda0;
var converged = false;

for (var i = 0; i < 20; i++) {
    lambda0 = lambda;

    var sinlambda = Math.sin(lambda);
    var coslambda = Math.cos(lambda);

    // eq. 14
    var sin2sigma = (cosU2 * sinlambda * cosU2 * sinlambda) + Math.pow(cosU1sinU2 - sinU1cosU2 *
                    coslambda, 2.0);
    var sinsigma = Math.sqrt(sin2sigma);

    // eq. 15
    var cossigma = sinU1sinU2 + (cosU1cosU2 * coslambda);

    // eq. 16
    sigma = Math.atan2(sinsigma, cossigma);

    // eq. 17    Careful!  sin2sigma might be almost 0!
    var sinalpha = (sin2sigma == 0) ? 0.0 : cosU1cosU2 * sinlambda / sinsigma;
    var alpha = Math.asin(sinalpha);
    var cosalpha = Math.cos(alpha);
    var cos2alpha = cosalpha * cosalpha;

    // eq. 18    Careful!  cos2alpha might be almost 0!
    var cos2sigmam = cos2alpha == 0.0 ? 0.0 : cossigma - 2 * sinU1sinU2 / cos2alpha;
    var u2 = cos2alpha * a2b2b2;

    var cos2sigmam2 = cos2sigmam * cos2sigmam;

    // eq. 3
    A = 1.0 + u2 / 16384 * (4096 + u2 * (-768 + u2 * (320 - 175 * u2)));

    // eq. 4
    B = u2 / 1024 * (256 + u2 * (-128 + u2 * (74 - 47 * u2)));

    // eq. 6
    deltasigma = B * sinsigma * (cos2sigmam + B / 4 * (cossigma * (-1 + 2 * cos2sigmam2) - B / 6 *
                 cos2sigmam * (-3 + 4 * sin2sigma) * (-3 + 4 * cos2sigmam2)));

    // eq. 10
    var C = f / 16 * cos2alpha * (4 + f * (4 - 3 * cos2alpha));
```

```
        // eq. 11 (modified)
        lambda = omega + (1 - C) * f * sinalpha * (sigma + C * sinsigma * (cos2sigmam + C * cossigma * (-1
                 + 2 * cos2sigmam2)));

        // see how much improvement we got
        var change = Math.abs((lambda - lambda0) / lambda);

        if ((i > 1) && (change < 0.0000000000001)) {
            converged = true;
            break;
        }
    }

    // eq. 19
    var s = b * A * (sigma - deltasigma);
    var alpha1;
    var alpha2;

    // didn't converge?  must be N/S
    if (!converged) {
        if (phi1 > phi2) {
            alpha1 = 180.0;
            alpha2 = 0.0;
        }
        else (phi1 < phi2)
        {
            alpha1 = 0.0;
            alpha2 = 180.0;
        }
    }
    // else, it converged, so do the math
    else {
        var radians;
        // eq. 20
        radians = Math.atan2(cosU2 * Math.sin(lambda), (cosU1sinU2 - sinU1cosU2 * Math.cos(lambda)));
        if (radians < 0.0) radians += TwoPi;
        alpha1 = radians * 180.0 / Math.PI;

        // eq. 21
        radians = Math.atan2(cosU1 * Math.sin(lambda), (-sinU1cosU2 + cosU1sinU2 * Math.cos(lambda))) +
                  Math.PI;
        if (radians < 0.0) radians += TwoPi;
        alpha2 = radians * 180.0 / Math.PI;
    }

    if (alpha1 >= 360.0) alpha1 -= 360.0;
    if (alpha2 >= 360.0) alpha2 -= 360.0;

    return new GeodeticCurve(s, alpha1, alpha2, lat1, long1, lat2, long2, 0, 0);

}

// Determines the lat and long bounds for google maps
function DetermineLatLongBounds(maxLat, fromLong, minLat, toLong) {
    var left = 0.0;
    var right = 0.0;
    if (fromLong < toLong) {
        left = fromLong;
        right = toLong;
    }
    else {
        left = toLong;
        right = fromLong;
    }

    return new google.maps.LatLngBounds(new google.maps.LatLng(minLat, left), new
                                        google.maps.LatLng(maxLat, right));
}
```

```
// Calculates the lat long for the midpint of a route
function calculateGreatCircleMidpoint(fromLat, fromLong, toLat, toLong) {

    var dLat = (toLat - fromLat) * Math.PI / 180;
    var dLon = (toLong - fromLong) * Math.PI / 180;
    var lat1 = fromLat * Math.PI / 180;
    var lat2 = toLat * Math.PI / 180;
    var lon1 = fromLong * Math.PI / 180;

    var Bx = Math.cos(lat2) * Math.cos(dLon);
    var By = Math.cos(lat2) * Math.sin(dLon);
    var lat3 = Math.atan2(Math.sin(lat1) + Math.sin(lat2),
                      Math.sqrt((Math.cos(lat1) + Bx) * (Math.cos(lat1) + Bx) + By * By));
    var lon3 = lon1 + Math.atan2(By, Math.cos(lat1) + Bx);

    lat3 *= 180 / Math.PI;
    lon3 *= 180 / Math.PI;

    var ns, ew;

    if (lat3 >= 0)
        ns = "N";
    else
        ns = "S";

    if (lon3 >= 0)
        ew = "E";
    else
        ew = "W";

    var latLong = ns + Math.abs(lat3).toFixed(0) + ew + Math.abs(lon3).toFixed(0);

    return latLong;
}

//Determine the maximum gross takeoff weight for a given pavement and aircraft type
function PCNtoMaxGrossWeight(selectMDS, PCN)
{
    var numPCN = PCN.substring(0,3);
    var pavement = PCN.substring(3,5);
    var maxGrossTakeoffWeight = null;

    //C-5 pavement Regression Coefficients
    var c5_maxGrossTakeoffWeight = 769.0;
    var c5_FA_Beta0 = 201.41;
    var c5_FA_Beta1 = 17.26;
    var c5_FB_Beta0 = 172.07;
    var c5_FB_Beta1 = 15.53;
    var c5_FC_Beta0 = 159.89;
    var c5_FC_Beta1 = 12.59;
    var c5_FD_Beta0 = 174.29;
    var c5_FD_Beta1 = 8.32;
    var c5_RA_Beta0 = 196.48;
    var c5_RA_Beta1 = 22.19;
    var c5_RB_Beta0 = 15.54;
    var c5_RB_Beta1 = 35.85;
    var c5_RC_Beta0 = 190.93;
    var c5_RC_Beta1 = 16.64;
    var c5_RD_Beta0 = 182.12;
    var c5_RD_Beta1 = 13.71;

    //C-17 pavement Regression Coefficients
    var c17_maxGrossTakeoffWeight = 585.0;
    var c17_FA_Beta0 = 121.5882353;
    var c17_FA_Beta1 = 8.911764706;
    var c17_FB_Beta0 = 126.6153846;
    var c17_FB_Beta1 = 7.769230769;
    var c17_FC_Beta0 = 145.9591837;
```

```
var c17_FC_Beta1 = 6.183673469;
var c17_FD_Beta0 = 153.4545455;
var c17_FD_Beta1 = 4.590909091;
var c17_RA_Beta0 = 59.80;
var c17_RA_Beta1 = 10.10;
var c17_RB_Beta0 = 59.80;
var c17_RB_Beta1 = 10.10;
var c17_RC_Beta0 = 59.80;
var c17_RC_Beta1 = 10.10;
var c17_RD_Beta0 = 123.9130435;
var c17_RD_Beta1 = 6.586956522;

//C-130 pavement Regression Coefficients
var c130_maxGrossTakeoffWeight = 153.7;
var c130_FA_Beta0 = 62.50;
var c130_FA_Beta1 = 3.75;
var c130_FB_Beta0 = 57.31;
var c130_FB_Beta1 = 3.46;
var c130_FC_Beta0 = 46.92;
var c130_FC_Beta1 = 3.46;
var c130_FD_Beta0 = 41.55;
var c130_FD_Beta1 = 3.10;
var c130_RA_Beta0 = 57.31;
var c130_RA_Beta1 = 3.46;
var c130_RB_Beta0 = 56.07;
var c130_RB_Beta1 = 3.21;
var c130_RC_Beta0 = 52.00;
var c130_RC_Beta1 = 3.00;
var c130_RD_Beta0 = 50.16;
var c130_RD_Beta1 = 2.90;


switch (selectMDS) {
    case C5:
        switch (pavement) {
            case "FA":
                maxGrossTakeoffWeight = c5_FA_Beta0 + c5_FA_Beta1 * numPCN;
                break;
            case "FB":
                maxGrossTakeoffWeight = c5_FB_Beta0 + c5_FB_Beta1 * numPCN;
                break;
            case "FC":
                maxGrossTakeoffWeight = c5_FC_Beta0 + c5_FC_Beta1 * numPCN;
                break;
            case "FD":
                maxGrossTakeoffWeight = c5_FD_Beta0 + c5_FD_Beta1 * numPCN;
                break;
            case "RA":
                maxGrossTakeoffWeight = c5_RA_Beta0 + c5_RA_Beta1 * numPCN;
                break;
            case "RB":
                maxGrossTakeoffWeight = c5_RB_Beta0 + c5_RB_Beta1 * numPCN;
                break;
            case "RC":
                maxGrossTakeoffWeight = c5_RC_Beta0 + c5_RC_Beta1 * numPCN;
                break;
            case "RD":
                maxGrossTakeoffWeight = c5_RD_Beta0 + c5_RD_Beta1 * numPCN;
                break;
            default:
                maxGrossTakeoffWeight = c5_maxGrossTakeoffWeight;
                break;
        }
        break;
    case C17:
        switch (pavement) {
            case "FA":
                maxGrossTakeoffWeight = c17_FA_Beta0 + c17_FA_Beta1 * numPCN;
                break;
```

```
        case "FB":
            maxGrossTakeoffWeight = c17_FB_Beta0 + c17_FB_Beta1 * numPCN;
            break;
        case "FC":
            maxGrossTakeoffWeight = c17_FC_Beta0 + c17_FC_Beta1 * numPCN;
            break;
        case "FD":
            maxGrossTakeoffWeight = c17_FD_Beta0 + c17_FD_Beta1 * numPCN;
            break;
        case "RA":
            maxGrossTakeoffWeight = c17_RA_Beta0 + c17_RA_Beta1 * numPCN;
            break;
        case "RB":
            maxGrossTakeoffWeight = c17_RB_Beta0 + c17_RB_Beta1 * numPCN;
            break;
        case "RC":
            maxGrossTakeoffWeight = c17_RC_Beta0 + c17_RC_Beta1 * numPCN;
            break;
        case "RD":
            maxGrossTakeoffWeight = c17_RD_Beta0 + c17_RD_Beta1 * numPCN;
            break;
        default:
            maxGrossTakeoffWeight = c17_maxGrossTakeoffWeight;
            break;
    }
    break;
case C130:
    switch (pavement) {
        case "FA":
            maxGrossTakeoffWeight = c130_FA_Beta0 + c130_FA_Beta1 * numPCN;
            break;
        case "FB":
            maxGrossTakeoffWeight = c130_FB_Beta0 + c130_FB_Beta1 * numPCN;
            break;
        case "FC":
            maxGrossTakeoffWeight = c130_FC_Beta0 + c130_FC_Beta1 * numPCN;
            break;
        case "FD":
            maxGrossTakeoffWeight = c130_FD_Beta0 + c130_FD_Beta1 * numPCN;
            break;
        case "RA":
            maxGrossTakeoffWeight = c130_RA_Beta0 + c130_RA_Beta1 * numPCN;
            break;
        case "RB":
            maxGrossTakeoffWeight = c130_RB_Beta0 + c130_RB_Beta1 * numPCN;
            break;
        case "RC":
            maxGrossTakeoffWeight = c130_RC_Beta0 + c130_RC_Beta1 * numPCN;
            break;
        case "RD":
            maxGrossTakeoffWeight = c130_RD_Beta0 + c130_RD_Beta1 * numPCN;
            break;
        default:
            maxGrossTakeoffWeight = c130_maxGrossTakeoffWeight;
            break;
    }
    break;
default:
    switch (pavement) {
        case "FA":
            maxGrossTakeoffWeight = c17_FA_Beta0 + c17_FA_Beta1 * numPCN;
            break;
        case "FB":
            maxGrossTakeoffWeight = c17_FB_Beta0 + c17_FB_Beta1 * numPCN;
            break;
        case "FC":
            maxGrossTakeoffWeight = c17_FC_Beta0 + c17_FC_Beta1 * numPCN;
            break;
        case "FD":
```

```
                    maxGrossTakeoffWeight = c17_FD_Beta0 + c17_FD_Beta1 * numPCN;
                    break;
                case "RA":
                    maxGrossTakeoffWeight = c17_RA_Beta0 + c17_RA_Beta1 * numPCN;
                    break;
                case "RB":
                    maxGrossTakeoffWeight = c17_RB_Beta0 + c17_RB_Beta1 * numPCN;
                    break;
                case "RC":
                    maxGrossTakeoffWeight = c17_RC_Beta0 + c17_RC_Beta1 * numPCN;
                    break;
                case "RD":
                    maxGrossTakeoffWeight = c17_RD_Beta0 + c17_RD_Beta1 * numPCN;
                    break;
                default:
                    maxGrossTakeoffWeight = c17_maxGrossTakeoffWeight;
                    break;
            }
            break;
    }
    return maxGrossTakeoffWeight;
}

// Determines the average temperature given a latitude in degrees and an elevation in feet.
function DetermineAverageTemperature(latitude, elevation) {
    var currentDate = new Date();
    var currentMonth = currentDate.getMonth();

    // Jan is 0 switches to Jan is 1
    var month = currentMonth + 1;
    var airfieldAveTemp = 0.0;

    //Temperature regression Coefficents
    var Jan_Beta0 = 23.16;
    var Jan_Beta1 = -0.2832;
    var Jan_Beta2 = -0.0084;
    var Jan_Beta3 = 26.06;

    var Feb_Beta0 = 24.78;
    var Feb_Beta1 = -0.2575;
    var Feb_Beta2 = -0.0092;
    var Feb_Beta3 = 30.32;

    var Mar_Beta0 = 26.80;
    var Mar_Beta1 = -0.1651;
    var Mar_Beta2 = -0.0099;
    var Mar_Beta3 = 28.21;

    var Apr_Beta0 = 27.67;
    var Apr_Beta1 = -0.0311;
    var Apr_Beta2 = -0.0097;
    var Apr_Beta3 = 20.00;

    var May_Beta0 = 27.06;
    var May_Beta1 = 0.0825;
    var May_Beta2 = -0.0088;
    var May_Beta3 = 15.77;

    var Jun_Beta0 = 26.33;
    var Jun_Beta1 = 0.1696;
    var Jun_Beta2 = -0.0080;
    var Jun_Beta3 = 12.45;

    var Jul_Beta0 = 26.17;
    var Jul_Beta1 = 0.1984;
    var Jul_Beta2 = -0.0077;
    var Jul_Beta3 = 12.91;

    var Aug_Beta0 = 26.80;
```

```
var Aug_Beta1 = 0.1565;
var Aug_Beta2 = -0.0080;
var Aug_Beta3 = 18.38;

var Sep_Beta0 = 27.26;
var Sep_Beta1 = 0.0703;
var Sep_Beta2 = -0.0085;
var Sep_Beta3 = 22.54;

var Oct_Beta0 = 26.89;
var Oct_Beta1 = -0.0315;
var Oct_Beta2 = -0.0089;
var Oct_Beta3 = 21.87;

var Nov_Beta0 = 24.95;
var Nov_Beta1 = -0.1474;
var Nov_Beta2 = -0.0086;
var Nov_Beta3 = 20.51;

var Dec_Beta0 = 23.21;
var Dec_Beta1 = -0.2418;
var Dec_Beta2 = -0.0082;
var Dec_Beta3 = 22.29;

var Yr_Beta0 = 25.92;
var Yr_Beta1 = -0.0400;
var Yr_Beta2 = -0.0087;
var Yr_Beta3 = 20.94;

//Elevation regression coefficients
var elev_Beta1 = -1.9812;

//Calculate the latitude determined average monthly or yearly temperature
switch (month)
{
    case "1":
        airfieldAveTemp = Jan_Beta0 + Jan_Beta1 * latitude + Jan_Beta2 * Math.pow(latitude, 2) +
                          Jan_Beta3 * Math.pow(latitude, 3) / 1000000;
        break;
    case "2":
        airfieldAveTemp = Feb_Beta0 + Feb_Beta1 * latitude + Feb_Beta2 * Math.pow(latitude, 2) +
                          Feb_Beta3 * Math.pow(latitude, 3) / 1000000;
        break;
    case "3":
        airfieldAveTemp = Mar_Beta0 + Mar_Beta1 * latitude + Mar_Beta2 * Math.pow(latitude, 2) +
                          Mar_Beta3 * Math.pow(latitude, 3) / 1000000;
        break;
    case "4":
        airfieldAveTemp = Apr_Beta0 + Apr_Beta1 * latitude + Apr_Beta2 * Math.pow(latitude, 2) +
                          Apr_Beta3 * Math.pow(latitude, 3) / 1000000;
        break;
    case "5":
        airfieldAveTemp = May_Beta0 + May_Beta1 * latitude + May_Beta2 * Math.pow(latitude, 2) +
                          May_Beta3 * Math.pow(latitude, 3) / 1000000;
        break;
    case "6":
        airfieldAveTemp = Jun_Beta0 + Jun_Beta1 * latitude + Jun_Beta2 * Math.pow(latitude, 2) +
                          Jun_Beta3 * Math.pow(latitude, 3) / 1000000;
        break;
    case "7":
        airfieldAveTemp = Jul_Beta0 + Jul_Beta1 * latitude + Jul_Beta2 * Math.pow(latitude, 2) +
                          Jul_Beta3 * Math.pow(latitude, 3) / 1000000;
        break;
    case "8":
        airfieldAveTemp = Aug_Beta0 + Aug_Beta1 * latitude + Aug_Beta2 * Math.pow(latitude, 2) +
                          Aug_Beta3 * Math.pow(latitude, 3) / 1000000;
        break;
    case "9":
        airfieldAveTemp = Sep_Beta0 + Sep_Beta1 * latitude + Sep_Beta2 * Math.pow(latitude, 2) +
```

```
                            Sep_Beta3 * Math.pow(latitude, 3) / 1000000;
            break;
        case "10":
            airfieldAveTemp = Oct_Beta0 + Oct_Beta1 * latitude + Oct_Beta2 * Math.pow(latitude, 2) +
                            Oct_Beta3 * Math.pow(latitude, 3) / 1000000;
            break;
        case "11":
            airfieldAveTemp = Nov_Beta0 + Nov_Beta1 * latitude + Nov_Beta2 * Math.pow(latitude, 2) +
                            Nov_Beta3 * Math.pow(latitude, 3) / 1000000;
            break;
        case "12":
            airfieldAveTemp = Dec_Beta0 + Dec_Beta1 * latitude + Dec_Beta2 * Math.pow(latitude, 2) +
                            Dec_Beta3 * Math.pow(latitude, 3) / 1000000;
            break;
        default:
            airfieldAveTemp = Yr_Beta0 + Yr_Beta1 * latitude + Yr_Beta2 * Math.pow(latitude, 2) + Yr_Beta3
                            * Math.pow(latitude, 3) / 1000000;
            break;
    }

    //adjust airfield temperature for elevation
    airfieldAveTemp = airfieldAveTemp + elevation * elev_Beta1 / 1000;

    return airfieldAveTemp.toFixed(2);
}


// Returns the headwind component in knots
function HeadWind(rwyHdg, windDirection, windSpeed) {

    if (typeof windDirection === "undefined") {
        headwnd = - windSpeed / 1.0;
    }
    else {
        var windAngle = windDirection - rwyHdg;

        if (windAngle > 180)
            windAngle = -360 + windAngle;
        if (windAngle <= -180)
            windAngle = 360 + windAngle;
        if ((windAngle >= -90 && windAngle < 0) || (windAngle > 90 && windAngle <= 180))
            windAngle = -windAngle;
        var headwnd = windSpeed * Math.cos(windAngle * Math.PI / 180.0)
    }

    return headwnd;
}

// Returns the crosswing component in knots
function CrossWind(rwyHdg, windDirection, windSpeed) {
    if (typeof windDirection === "undefined") {
        crosswnd = windSpeed / 1.0;
    }
    else {
        var windAngle = windDirection - rwyHdg;

        if (windAngle > 180)
            windAngle = -360 + windAngle;
        if (windAngle <= -180)
            windAngle = 360 + windAngle;
        var crosswnd = windSpeed * Math.sin(windAngle * Math.PI / 180.0)
    }

    return crosswnd;
}
```

```
// returns flightPath object that represents the shortest flight path from an array of flight paths
function SelectShortestFlightPath(fltPaths) {
    var shortestFltPath = new flightPath();

    shortestFltPath = fltPaths[0];
    for (var i = 1; i < fltPaths.length; i++) {
        if (fltPaths[i].totalDist < shortestFltPath.totalDist)
            shortestFltPath = fltPaths[i];
    }

    return shortestFltPath;
}
```

restrictedPolygon Functions

Name	Inputs	Description
DoesCurveIntersect	(geoCurveSortie)	Returns boolean true if geodeticCurve object intersects restrictedPolygon object.
CalculateIntersections	(geoCurve, crossID)	Returns Array of waypoints representing the intersections of the sortie and the restrictedPolygon object's geocurves. CrossID is boolean representing if sortie crosses the International Dateline.
AddWaypoint	(wpt)	Adds waypoint to restrictedPolygon object.
CenterWpt	()	Returns waypoint object representing center of polygon.

```
//determines if curve intersects restricted airspace
restrictedPolygon.prototype.DoesCurveIntersect = function (geoCurveSortie) {
    var crossID = false;
    var potIntersectCoords;
    var intersections;
    var wpts;

    // determine min and max latitudes for great circle path
    geoCurveSortie.LatMaxMin();

    //determine if longitude pair results in cross of 180E/W
    var deltaLong = Math.abs(geoCurveSortie.toLong - geoCurveSortie.fromLong);
    if (deltaLong > 180)
        crossID = true;

    // Determine if polygon potentially impacts sortie flight path
    // Is this restricted airspace block a potential factor for the sortie
    // if min lat of restricted airspace is < max lat of great circle or max lat of restricted airspace is
    // > min lat of great circle
    if (this.minLat < geoCurveSortie.maxLat || this.maxLat > geoCurveSortie.minLat) {
        //restricted airspace might potentially interfere due to vertical
        if (crossID) {
            if (this.minLong < Math.min(geoCurveSortie.fromLong, geoCurveSortie.toLong) ||
                this.maxLong > Math.max(geoCurveSortie.fromLong, geoCurveSortie.toLong)) {
                this.potentialImpact = true;
            }
            else {
                this.potentialImpact = false;
            }
        }
        else {
            if ((this.maxLong > Math.min(geoCurveSortie.fromLong, geoCurveSortie.toLong) &&
                 this.maxLong < Math.max(geoCurveSortie.fromLong, geoCurveSortie.toLong)) ||
                (this.minLong > Math.min(geoCurveSortie.fromLong, geoCurveSortie.toLong) &&
                 this.minLong < Math.max(geoCurveSortie.fromLong, geoCurveSortie.toLong))) {
                this.potentialImpact = true;
```

```
            }
            else {
                this.potentialImpact = false;
            }
        }
    }

    // Determine if restricted polygon actually impacts flight path
    if (this.potentialImpact) {
        // calculate intersections
        this.calculateIntersections(geoCurveSortie, crossID);

        // if number of intersections is greater than zero calculate set of alternative flight paths
        if (this.intersections.length > 0) {
            return true;
        }
        else
            return false;
    }
    else {
        return false;
    }
}

// Returns intersections array of waypoint objects that represent intersections
restrictedPolygon.prototype.CalculateIntersections = function (geoCurve, crossID) {
    var intersection;
    var intersections = new Array();
    for (var i = 0; i < this.geoCurves.length; i++) {
        intersection = this.geoCurves[i].calculateIntersection(geoCurve);
        if (intersection.lat < this.geoCurves[i].maxLat && intersection.lat > this.geoCurves[i].minLat &&
            intersection.lat < geoCurve.maxLat && intersection.lat > geoCurve.minLat) {
            if (crossID) {
                if (intersection.long < Math.max(this.geoCurves[i].fromLong, this.geoCurves[i].toLong) &&
                    intersection.long > Math.min(this.geoCurves[i].fromLong, this.geoCurves[i].toLong) &&
                   (intersection.long > Math.max(geoCurve.fromLong, geoCurve.toLong) ||
                    intersection.long < Math.min(geoCurve.fromLong, geoCurve.toLong))) {
                    intersections.push(intersection);
                }
            }
            else {
                if (intersection.long < Math.max(this.geoCurves[i].fromLong, this.geoCurves[i].toLong) &&
                    intersection.long > Math.min(this.geoCurves[i].fromLong, this.geoCurves[i].toLong) &&
                    intersection.long < Math.max(geoCurve.fromLong, geoCurve.toLong) &&
                    intersection.long > Math.min(geoCurve.fromLong, geoCurve.toLong)) {
                    intersections.push(intersection);
                }
            }
        }
    }
    this.intersections = intersections;
}


// adds a waypoint object to waypoints array property of restrictedPolygon object
restrictedPolygon.prototype.AddWaypoint = function (wpt) {
    this.waypoints.push(wpt)
    this.geoCurves = new Array();
    var geoCurve;
    this.maxLat = this.waypoints[0].lat;
    this.minLat = this.waypoints[0].lat;
    this.maxLong = this.waypoints[0].long;
    this.minLong = this.waypoints[0].long;
    if (this.waypoints.length > 1) {
        for (var i = 0; i < this.waypoints.length; i++) {
            if (i == this.waypoints.length - 1) {
                geoCurve = new VincentyDistance(this.waypoints[i].lat, this.waypoints[i].long,
                                                this.waypoints[0].lat, this.waypoints[0].long);
                geoCurve.LatMaxMin();
```

```
            }
            else {
                geoCurve = new VincentyDistance(this.waypoints[i].lat, this.waypoints[i].long,
                                                this.waypoints[i + 1].lat, this.waypoints[i + 1].long);
                geoCurve.LatMaxMin();
            }
            this.geoCurves.push(geoCurve);
            if (geoCurve.maxLat > this.maxLat)
                this.maxLat = geoCurve.maxLat;
            if (geoCurve.minLat < this.minLat)
                this.minLat = geoCurve.minLat;
            if (this.waypoints[i].long > this.maxLong)
                this.maxLong = this.waypoints[i].long;
            if (this.waypoints[i].long < this.minLong)
                this.minLong = this.waypoints[i].long;
        }
    }
    this.potentialImpact = false;
}

// returns the center waypoint between min and max lat and min and max long of restrictedPolygon object
restrictedPolygon.prototype.CenterWpt = function () {
    var lat = (this.maxLat + this.minLat) / 2;
    var long = (this.maxLong + this.minLong) / 2;
    var wpt = new waypoint(lat, long);

    return wpt;
}
```

waypoint Functions

Name	Inputs	Description
GcMapWpt	()	Returns the textual format of a lat-long pair for visualization on Great Circle Mapper website.

```
// returns the textual format for a lat-long pair for display on the great circle mapper web site
waypoint.prototype.GcMapWpt = function () {
    var GCmapWpt = "";
    var waypointNS = "";
    var waypointEW = "";

    if (this.lat >= 0)
        waypointNS = "N";
    else
        waypointNS = "S";

    if (this.long >= 0)
        waypointEW = "E";
    else
        waypointEW = "W";

    GCmapWpt = this.lat.toFixed(0) + waypointNS + this.long.toFixed(0) + waypointEW;
    return GCmapWpt;
}
```

GeodeticCurve Functions

Name	Inputs	Description
LatMaxMin	()	Returns the maximum and minimum latitude over the geodetic curve from origin to destination.
CalcLatGivenLong	(lon)	Returns the latitude given a longitude for the geodetic curve.
CalcLongGivenLat	(lat)	Returns the LongitudePair given a latitude for the geodetic curve.
CalculateIntersection	(geocurve)	Returns the waypoint object of the intersection of the two geocurves.
DetermineOptimalPath AroundRest	(restPoly)	Returns the flightPath object that optimally goes around the airspace of a restrictedPolygon object.
BuildAltFlightPaths	(restPoly)	Returns an array of possible flightPath objects that go around the airspace of a restrictedPolygon object.

```
// Calculates the maximum and minimum latitude given a geodetic curve
GeodeticCurve.prototype.LatMaxMin = function() {
    var az = this.azimuth;
    var revaz = this.reverseAzimuth;
    var latMxMn = 180.0 * Math.acos(Math.abs(Math.sin(az * Math.PI / 180.0) * Math.cos(this.fromLat *
                  Math.PI / 180.0))) / Math.PI;
    if ((az >= 0 && az < 90 && revaz >= 180 && revaz < 270) ||
        (az >= 90 && az < 180 && revaz >= 270 && revaz < 360) ||
        (az >= 180 && az < 270 && revaz >= 0 && revaz < 90) ||
        (az >= 270 && az < 360 && revaz >= 90 && revaz < 180))
    {
        if (this.fromLat > this.toLat) {
            this.maxLat = this.fromLat;
            this.minLat = this.toLat;
        }
        else {
            this.maxLat = this.toLat;
            this.minLat = this.fromLat;
        }
    }
    else
    {
        if (az > 90 && az < 270 && revaz > 90 && revaz < 270) {
            this.minLat = -latMxMn;
            if (this.fromLat > this.toLat) {
                this.maxLat = this.fromLat;
            }
            else {
                this.maxLat = this.toLat;
            }
        }
        else {
            this.maxLat = latMxMn;
            if (this.fromLat > this.toLat) {
                this.minLat = this.toLat;
            }
            else {
                this.minLat = this.fromLat;
            }
        }
    }
}

// Returns the latitude given a geodetic curve and a longitude
GeodeticCurve.prototype.CalcLatGivenLong = function (lon) {
    var alpha0; // the bearing angle at the equator for the great circle in radians
```

```
    var lambda0; // the longitude where the great circle crosses the equator in radians
    var phi1; // the latitude of the selected origin on the geocurve in radians
    var lambda1; // the longitude for the selected origin on the geocurve in radians
    var alpha1; // the bearing for the latitude selected in radians
    var sigma01; // the angle from great circle point on equator to origin
    var sigma02; // the angle from great circle point on equator to selected point
    var lambda2; // the longitude that determines the latitude
    var phi2; // the latitude that is being calculated
    var lat; // the latitude in degrees

    alpha1 = this.azimuth * Math.PI / 180.0; // bearing of origin airfield
    phi1 = this.fromLat * Math.PI / 180.0; // latitude of origin airfield
    lambda1 = this.fromLong * Math.PI / 180.0; // longitude of origin airfield
    lambda2 = lon * Math.PI / 180.0; // longitude of input point
    // determine bearin angle at the equator
    alpha0 = Math.atan2(Math.sin(alpha1) * Math.cos(phi1), Math.sqrt(Math.pow(Math.cos(alpha1), 2) +
             Math.pow(Math.sin(alpha1), 2) * Math.pow(Math.sin(phi1), 2)));

    // determine angle from great circle point on equator to origin airfield
    sigma01 = Math.atan2(Math.tan(phi1), Math.cos(alpha1));

    // determine the longitude at the equator
    lambda0 = lambda1 - Math.atan2(Math.sin(alpha0) * Math.sin(sigma01), Math.cos(sigma01));

    // determine angle from great circle point on equator to selected point at input longitude
    var deltaLambda = lambda2 - lambda0;
    if (alpha0 > 0) {
        if ((deltaLambda > Math.PI / 2 && deltaLambda < 3 * Math.PI / 2) || (deltaLambda < -Math.PI / 2 &&
             deltaLambda > -3 * Math.PI / 2)) {
            sigma02 = -Math.atan2(Math.tan(deltaLambda), Math.sin(alpha0));
        }
        else {
            sigma02 = Math.atan2(Math.tan(deltaLambda), Math.sin(alpha0));
        }

    }
    else {
        if ((deltaLambda < Math.PI / 2 && deltaLambda > -Math.PI / 2) || (deltaLambda > 3 * Math.PI / 2)
            || (deltaLambda < -3 * Math.PI / 2)) {
            sigma02 = -Math.atan2(Math.tan(deltaLambda), Math.sin(alpha0));
        }
        else {
            sigma02 = Math.atan2(Math.tan(deltaLambda), Math.sin(alpha0));
        }
    }

    // determine latitude of point
    phi2 = Math.asin(Math.cos(alpha0) * Math.sin(sigma02));
    lat = phi2 * 180.0 / Math.PI;
    debugTxt2 += "<tr><td>" + lon + "</td><td>" + lat + "</td><td>" + alpha0 * 180 / Math.PI + "</td><td>"
                 + sigma02 * 180 / Math.PI + "</td><td>" +
                 deltaLambda * 180 / Math.PI + "</td><td>" + Math.sin(alpha0) + "</td><td>" +
                 Math.tan(deltaLambda) + "</td></tr>";
    return lat;
}

// Returns a longitude pair given a latitude
GeodeticCurve.prototype.CalcLongGivenLat = function (lat) {
    var alpha0; // the bearing angle at the equator for the great circle in radians
    var lambda0; // the longitude where the great circle crosses the equator in radians
    var phi1; // the latitude of the selected origin on the geocurve in radians
    var lambda1; // the longitude for the selected origin on the geocurve in radians
    var alpha1; // the bearing for the latitude selected in radians
    var sigma01; // the angle from great circle point on equator to origin
    var sigma02; // the angle from great circle point on equator to selected point
    var sigma03; // the angle from great circle point on equator to selected point
    var phi2; // the latitude that determines the longitude
    var lambda2; // the first longitude for selected latitude
    var lambda3; // the second longitude for selected latitude
```

```
    var longPair; // the longitude pair in degrees


    alpha1 = this.azimuth * Math.PI / 180.0; // bearing of origin airfield
    phi1 = this.fromLat * Math.PI / 180.0; // latitude of origin airfield
    lambda1 = this.fromLong * Math.PI / 180.0; // longitude of origin airfield
    phi2 = lat * Math.PI / 180.0; // latitude of input point

    // determine bearin angle at the equator
    alpha0 = Math.atan2(Math.sin(alpha1) * Math.cos(phi1), Math.sqrt(Math.pow(Math.cos(alpha1), 2) +
             Math.pow(Math.sin(alpha1), 2) * Math.pow(Math.sin(phi1), 2)));

    // determine angle from great circle point on equator to origin airfield
    sigma01 = Math.atan2(Math.tan(phi1), Math.cos(alpha1));

    // determine the longitude at the equator
    lambda0 = lambda1 - Math.atan2(Math.sin(alpha0) * Math.sin(sigma01), Math.cos(sigma01));

    // determine angle from great circle point on equator to selected point at input latitude
    sigma02 = Math.asin(Math.sin(phi2) / Math.cos(alpha0));

    // determine longitudes of points
    lambda2 = lambda0 + Math.atan2(Math.sin(alpha0) * Math.sin(sigma02), Math.cos(sigma02));
    lambda3 = lambda0 + Math.atan2(Math.sin(alpha0) * Math.sin(Math.PI - sigma02), Math.cos(Math.PI –
              sigma02));
    longPair = new LongitudePair(lambda2 * 180 / Math.PI, lambda3 * 180 / Math.PI);

    return longPair;
}

// Returns a possible intersection lat-long pair given two geocurves
GeodeticCurve.prototype.CalculateIntersection = function (geoCurve) {
    var intersection = new waypoint();

    lat1 = this.fromLat * Math.PI / 180;
    lon1 = this.fromLong * Math.PI / 180;
    lat2 = geoCurve.fromLat * Math.PI / 180;
    lon2 = geoCurve.fromLong * Math.PI / 180;

    brng13 = this.azimuth * Math.PI / 180;
    brng23 = geoCurve.azimuth * Math.PI / 180;
    dLat = lat2 - lat1;
    dLon = lon2 - lon1;

    dist12 = 2 * Math.asin(Math.sqrt(Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                Math.cos(lat1) * Math.cos(lat2) * Math.sin(dLon / 2) * Math.sin(dLon / 2)));
    if (dist12 == 0) return null;

    // initial/final bearings between points
    brngA = Math.acos((Math.sin(lat2) - Math.sin(lat1) * Math.cos(dist12)) /
                      (Math.sin(dist12) * Math.cos(lat1)));
    if (isNaN(brngA)) brngA = 0;  // protect against rounding
    brngB = Math.acos((Math.sin(lat1) - Math.sin(lat2) * Math.cos(dist12)) /
                      (Math.sin(dist12) * Math.cos(lat2)));

    if (Math.sin(lon2 - lon1) > 0) {
        brng12 = brngA;
        brng21 = 2 * Math.PI - brngB;
    } else {
        brng12 = 2 * Math.PI - brngA;
        brng21 = brngB;
    }

    alpha1 = (brng13 - brng12 + Math.PI) % (2 * Math.PI) - Math.PI;  // angle 2-1-3
    alpha2 = (brng21 - brng23 + Math.PI) % (2 * Math.PI) - Math.PI;  // angle 1-2-3

    if (Math.sin(alpha1) == 0 && Math.sin(alpha2) == 0) return null;  // infinite intersections
    //if (Math.sin(alpha1) * Math.sin(alpha2) < 0) return null;       // ambiguous intersection
```

```
    alpha3 = Math.acos(-Math.cos(alpha1) * Math.cos(alpha2) +
                       Math.sin(alpha1) * Math.sin(alpha2) * Math.cos(dist12));
    dist13 = Math.atan2(Math.sin(dist12) * Math.sin(alpha1) * Math.sin(alpha2),
                        Math.cos(alpha2) + Math.cos(alpha1) * Math.cos(alpha3))
    lat3 = Math.asin(Math.sin(lat1) * Math.cos(dist13) +
                     Math.cos(lat1) * Math.sin(dist13) * Math.cos(brng13));
    dLon13 = Math.atan2(Math.sin(brng13) * Math.sin(dist13) * Math.cos(lat1),
                        Math.cos(dist13) - Math.sin(lat1) * Math.sin(lat3));
    lon3 = lon1 + dLon13;

    lon3 = (lon3 + 3 * Math.PI) % (2 * Math.PI) - Math.PI;  // normalise to -180..+180º

    intersection.lat = lat3 * 180 / Math.PI;
    intersection.long = lon3 * 180 / Math.PI;

    return intersection;
}

// returns the best flight path around a restricted area given a sortie that penetrates a restricted area
GeodeticCurve.prototype.DetermineOptimalPathAroundRest = function (restPoly) {
    var fltPaths;
    var optFltPath;
    //calculate the set of flight paths
    fltPaths = this.buildAltFlightPaths(restPoly);

    //select the flight path with the minimal distance
    optFltPath = selectShortestFlightPath(fltPaths);

    return optFltPath;
}

// Returns an array of possible flight paths around a restrictedPolygon object
GeodeticCurve.prototype.BuildAltFlightPaths = function (restPoly) {
    var fltPath, fltPath1, fltPath2;
    var fltSegments = new Array();
    var fltPaths = new Array();
    var fltSegOrigin;
    var fltSegDest;
    var fltSegInt;
    var geoCurvesOriginWpt = new Array();
    var geoCurvesWptDest = new Array();
    var geoCurvesOriginCtr;
    var geoCurvesCtrDest;
    var totalDist, geoCurvesWptToWpt;
    var maxAngleDiff = -180;
    var minAngleDiff = 180;
    var maxAngleDiffRev = -180;
    var minAngleDiffRev = 180;
    var maxOriginWptAz, minOriginWptAz;
    var maxWptDestRevAz, minWptDestRevAz;

    // calculate center of restricted area
    var centerWpt = restPoly.centerWpt();

    // create geodetic curve origin to center of restricted
    geoCurvesOriginCtr = VincentyDistance(this.fromLat, this.fromLong, centerWpt.lat, centerWpt.long);

    // create geodetic curve center of restricted to dest
    geoCurvesCtrDest = VincentyDistance(centerWpt.lat, centerWpt.long, this.toLat, this.toLong);

    for (var i = 0; i < restPoly.waypoints.length; i++) {


        // create geodetic curve origin to wpt
        geoCurvesOriginWpt[i] = VincentyDistance(this.fromLat, this.fromLong, restPoly.waypoints[i].lat,
                                restPoly.waypoints[i].long);
        geoCurvesOriginWpt[i].LatMaxMin();

        //determine angular difference origin to center vs origin to waypoint
```

```
        var angleDiffFromCtr = geoCurvesOriginWpt[i].azimuth - geoCurvesOriginCtr.azimuth;
        if (angleDiffFromCtr > 180) {
            angleDiffFromCtr -= 360;
        }
        else if (angleDiffFromCtr < -180) {
            angleDiffFromCtr += 360;
        }

        //determine if wpt has the max or min azimuth
        if (angleDiffFromCtr > maxAngleDiff) {
            maxOriginWptAz = geoCurvesOriginWpt[i].azimuth;
            maxAngleDiff = angleDiffFromCtr;
        }
        if (angleDiffFromCtr < minAngleDiff) {
            minOriginWptAz = geoCurvesOriginWpt[i].azimuth;
            minAngleDiff = angleDiffFromCtr;
        }

        // create geodetic curves wpt to destination
        geoCurvesWptDest[i] = VincentyDistance(restPoly.waypoints[i].lat, restPoly.waypoints[i].long,
                                               this.toLat, this.toLong);
        geoCurvesWptDest[i].LatMaxMin();

        //determine angular difference center to destination vs waypoint to destination
        var angleDiffFromCtr = geoCurvesWptDest[i].reverseAzimuth - geoCurvesCtrDest.reverseAzimuth;
        if (angleDiffFromCtr > 180) {
            angleDiffFromCtr -= 360;
        }
        else if (angleDiffFromCtr < -180) {
            angleDiffFromCtr += 360;
        }

        //determine if wpt has the max or min azimuth
        if (angleDiffFromCtr > maxAngleDiffRev) {
            maxWptDestRevAz = geoCurvesWptDest[i].reverseAzimuth;
            maxAngleDiffRev = angleDiffFromCtr;
        }
        if (angleDiffFromCtr < minAngleDiffRev) {
            minWptDestRevAz = geoCurvesWptDest[i].reverseAzimuth;
            minAngleDiffRev = angleDiffFromCtr;
        }
    }

    for (var i = 0; i < restPoly.waypoints.length; i++) {
        // determine if selected waypoint has max or min azimuth from origin
        if (geoCurvesOriginWpt[i].azimuth == maxOriginWptAz || geoCurvesOriginWpt[i].azimuth ==
            minOriginWptAz) {
            // build flight segment origin to wpt
            fltSegOrigin = new flightSegment(this.fromLat, this.fromLong, restPoly.waypoints[i].lat,
                                restPoly.waypoints[i].long, geoCurvesOriginWpt[i].distance / 1852.0);

            // begin building flight path
            fltSegments = [];
            fltSegments.push(fltSegOrigin);

            if (geoCurvesWptDest[i].reverseAzimuth == maxWptDestRevAz ||
                geoCurvesWptDest[i].reverseAzimuth == minWptDestRevAz) {
                // build flight segment wpt to dest
                fltSegDest = new flightSegment(restPoly.waypoints[i].lat, restPoly.waypoints[i].long,
                                               this.toLat, this.toLong, geoCurvesWptDest[i].distance / 1852.0);

                // add destination leg
                fltSegments.push(fltSegDest);
                totalDist = fltSegOrigin.dist + fltSegDest.dist;
                fltPath = new flightPath(fltSegments, totalDist);
                fltPaths.push(fltPath);
            }
            else {
                if (geoCurvesOriginWpt[i].azimuth == maxOriginWptAz) {
```

```
                    // find wpt with min reverse az
                    for (var j = 0; j < restPoly.waypoints.length; j++) {
                        if (i != j) {
                            if (geoCurvesWptDest[j].reverseAzimuth == minWptDestRevAz) {
                                // build geocurve for wpt1 to wpt2
                                geoCurvesWptToWpt = VincentyDistance(restPoly.waypoints[i].lat,
                                                                     restPoly.waypoints[i].long,
                                                                     restPoly.waypoints[j].lat,
                                                                     restPoly.waypoints[j].long);
                                // build flight segment wpt1 to wpt2
                                fltSegInt = new flightSegment(restPoly.waypoints[i].lat,
                                                              restPoly.waypoints[i].long,
                                                              restPoly.waypoints[j].lat,
                                                              restPoly.waypoints[j].long,
                                                              geoCurvesWptToWpt.distance / 1852.0);

                                // build flight segment wpt to dest
                                fltSegDest = new flightSegment(restPoly.waypoints[j].lat,
                                                       restPoly.waypoints[j].long, this.toLat, this.toLong,
                                                                geoCurvesWptDest[j].distance / 1852.0);

                                // add destination leg
                                fltSegments.push(fltSegInt);
                                fltSegments.push(fltSegDest);
                                totalDist = fltSegOrigin.dist + fltSegInt.dist + fltSegDest.dist;
                                fltPath = new flightPath(fltSegments, totalDist);
                                fltPaths.push(fltPath);
                            }
                        }
                    }
                }
                else {
                    // find wpt with max reverse az
                    for (var j = 0; j < restPoly.waypoints.length; j++) {
                        if (i != j) {
                            if (geoCurvesWptDest[j].reverseAzimuth == maxWptDestRevAz) {
                                // build geocurve for wpt1 to wpt2
                                geoCurvesWptToWpt = VincentyDistance(restPoly.waypoints[i].lat,
                                                                     restPoly.waypoints[i].long,
                                                                     restPoly.waypoints[j].lat,
                                                                     restPoly.waypoints[j].long);
                                // build flight segment wpt1 to wpt2
                                fltSegInt = new flightSegment(restPoly.waypoints[i].lat,
                                                              restPoly.waypoints[i].long,
                                                              restPoly.waypoints[j].lat,
                                                              restPoly.waypoints[j].long,
                                                              geoCurvesWptToWpt.distance / 1852.0);

                                // build flight segment wpt to dest
                                fltSegDest = new flightSegment(restPoly.waypoints[j].lat,
                                                       restPoly.waypoints[j].long, this.toLat, this.toLong,
                                                                geoCurvesWptDest[j].distance / 1852.0);

                                // add destination leg
                                fltSegments.push(fltSegInt);
                                fltSegments.push(fltSegDest);
                                totalDist = fltSegOrigin.dist + fltSegInt.dist + fltSegDest.dist;
                                fltPath = new flightPath(fltSegments, totalDist);
                                fltPaths.push(fltPath);
                            }
                        }
                    }
                }
            }
        }
    }

    return fltPaths;
}
```

Bibliography

Ai, T. J., & Kachitvichyanukul, V. (2009). A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research, 36*(5), 1693-1702.

Air Force. (2010). *Techincal Order 1C-5A-1-1.* Department of Defense.

Air Force. (2010). *Technical Order 1C-17A-1.* Department of Defense.

Air Force. (2010). *Technical Order 1C-5A-1.* Department of Defense.

Air Force. (2011). *Techincal Order 1C-130J-1.* Department of Defense.

Air Force. (2011). *Technical Order 1C-130J-1-1.* Department of Defense.

Air Force. (2011). *Technical Order 1C-17A-1.* Department of Defense.

(2011). *Air Force Instruction 11-2MDS Volume 3.* Scott AFB, IL: HQ AMC/A3V.

(2011). *Air Force Pamphlet 10-1403 Air Mobility Planning Factors.* Department of Defense.

Alexander, D. R., & Hall, J. W. (1991). ACN-PCN concepts for airport pavement management. *Aircraft/Pavement Interaction@ sAn Integrated System*, 393-405.

Babikian, R., Lukachko, S. P., & Waitz, I. A. (2001). Historical Fuel Efficiency Characteristics of Regional Aircraft from Technological, Operational and Cost Perspectives. *Journal of Air Transport Management, 8*(6), 389-400.

Baker, S., Morton, D., Rosenthal, R., & Williams, L. (2002). Optimizing Military Airlift. *Operations Research, 50*(4), 582-602.

Balakrishnan, A., Chien, T., & Wong, R. (1989). *Selecting Aircraft Routes for Long-haul Operations: A Formulation and Solution Method.* MIT Sloan.

Balas, E., & Padberg, M. W. (1972). On the Set Covering Problem. *Operations Research, 20*(6), 1152-1161.

Balinski, M. L., & Quandt, R. E. (1964). On an Interger Program for a Delivery Problem. *Operations Research, 12*(2), 300-304.

Barnes, J. W., Wiley, V. D., Moore, J. T., & Ryer, D. M. (2004). Solving the Aerial Fleet Refueling Problem using Group Theoretic Tabu Search. *Mathematical and Computer Modelling*, 617-640.

Barney, J. (1986). Organizational culture: can it be a source of sustained competitive advantage? *Academy of Management Review*, 656-665.

Barney, J. (1991). Firm Resources and Sustained Competitive Advantage. *Journal of Management, 17*(1), 99-120.

Becker, M. A., & Smith, S. F. (2000). *Mixed-Initiative Resource Management: The AMC Barrel Allocator.*

Bell, J. E., & McMullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics, 18*(1), 41-48.

Bodin, L. (1990). Twenty Years of Routing and Scheduling. *Operations Research, 38*(4), 571-581.

Brigantic, R. T., & Merrill, D. (2004). Algebra of Airlift. *Mathematical and Computer Modelling*, 649-656.

Burke, J. F., Love, R. J., & Macal, C. M. (2004). Modelling Force Deployments from Army Installations using the Transportation System Capability (TRANSCAP) Model: A Standardized Approach. *Mathematical and Computer Modelling*, 733-744.

Burstein, M., Ferguson, G., & Allen, J. (2003). *Integrating Agent-Based Mixed-Initiative Control With An Existing Multi-Agent Planning System.* DARPA.

CFR. (2010). *Title 14: Aeronautics and Space.* Office of the Secretary, Department of Transportation, Sec 19-5.

Chang, T. S. (2008). Best routes selection in international intermodal networks. *Computers & Operations Research, 35*(9), 2877-2891.

Clay, J. D. (1989). Temporal Constraint Propagation For Airlift Planning Analysis. *Air Force Institute of Technology Thesis*, 1-122.

Crino, J., Moore, J., Barnes, J., & Nanry, W. (2004). Solving the Theater Distribution Vehicle Routing and Scheduling Problem Using Group Theoretic Tabu Search. *Mathematical and Computer Modeling*, 599-616.

Crum, M. R., & Morrow, P. C. (2002). The influence of carrier scheduling practices on truck driver fatigue. *Transportation Journal*, 20-41.

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science, 6*(1), 80-91.

(2011). *Digital Aeronautical Flight Information File.* National Geospatial Intelligence Agency.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 269-271.

Dyer, J. S., & Sarin, R. K. (1979). Measurable multi-attribute value functions. *Opertions Research*, 810-822.

Eisenhardt, K. M., & Martin, J. A. (2000). Dynamic Capabilities: What are they? *Strategic Management Journal, 21*(10), 1105-1121.

FAA. (1996). *FAA Federal Aviation Regulations Part 121 Section 481.*

Ferguson, A. R., & Dantzig, G. B. (1954). *The Problem of Routing Aircraft, a Mathematical Solution.* (No. P-561). RAND CORP SANTA MONICA CALIF.

Flood, M. M. (1956). The Traveling-Salesman Problem. *Operations Research, 4*(1), 61-75.

Gagnepain, P., & Marin, P. (2007). The Effects of Airline Alliances: What do the Aggregate Data Say? *Journal of the Spanish Economic Association, 1*(3), 251-276.

Gavaghan, M. (2013). Retrieved from Mike Gavaghan Blog: http://www.gavaghan.org/blog/free-source-code/geodesy-library-vincentys-formula/

Gendreau, M., & Soriano, P. (1998). Airport pavement management systems: An appraisal of existing methodologies. *Transportation Research Part A: Policy and Practice*, 197-214.

Gill, M. M. (2005). Output Analysis and Comparison of Deployment Models with Varying Fidelity. *Air Force Institute of Technology Thesis*, 1-91.

Glover, F., & McMillan, C. (1986). The General Employee Scheduling Problem: An Integration of MS and AI. *Computer and Operations Research, 13*(5), 563-573.

Google Maps. (2013). Retrieved from https://developers.google.com/maps/

Great Circle Mapper. (2013). Retrieved from http://www.gcmap.com

Gueret, C., Jussien, N., Lhomme, O., Pavageau, C., & Prins, C. (2003). Loading aircraft for military operations. *Journal of the Operational Research Society, 54*(5), 458-465.

Hartlage, R. (2012). Rough-Cut Capacity Planning in Multimodal Freight Transportation Networks. *Air Force Institute of Technology Thesis*, 1-110.

Hatch, M. (1993). The dynamics of organizational culture. *Academy of Management, 18*(4), 657-693.

Held, M., & Karp, R. (1962). A Dynamic Programming Approach to Sequencing Problems. *Journal of the Society for Industrial and Applied Mathematics, 10*(1), 196-210.

Hileman, J., Katz, J., Mantilla, J., & Fleming, G. (2008). Payload Fuel Energy Efficiency as a Metric for Aviation Environmental Performance. *ICAS 2008 Proceedings.*

Jackson, J. A., Kloeber, J. M., Ralston, B. E., & Deckro, R. F. (1999). Selecting a portfolio of technologies: an application of decision analysis. *Decision Sciences*, 217=238.

Jackson, J., Jones, B., & Lehmkuhl, L. (1996). *An Operational Analysis for Air Force 2025: An Application of Value-Focused Thinking to Future Air and Space Capabilities.* Air Command and Staff College.

Jin, J., Crainic, T. G., & Lokketangen, A. (2012). A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems. *European Journal of Operational Research*.

(2009). *Joint Publication 3-17 Air Mobility Operations.* Department of Defense.

Karmarkar, N. (1984). A new polynomial time algorithm for linear programming. *Combinatorica, 4*(4), 373-395.

Keeney, R. L. (1994). Creativity in Decision making with Value Focused Thinking. *Sloan Management Review*, 33-41.

Koepke, C. G., Armacost, A. P., Barnhart, C., & Kolitz, S. E. (2008). An integer programming approach to support the US Air Force's air mobility network. *Computers & Operations Research, 35*(6), 1771-1788.

Koskosidis, Y. A., Powell, W. B., & Solomon, M. M. (1992). An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation Science, 26*(2), 69-85.

Kress, M., & Golany, B. (1994). Optimizing the Assignment of Aircrews to Aircraft in an Airlift Operation. *European Journal of Operational Research, 77*(3), 475-485.

Lahiri, K., Stekler, H. O., Yao, W. W., & Young, P. (2003). *Monthly Output Index for the U.S. Transportation Sector.* Suny: University of Albany, Department of Economics.

Lambert, G. R. (2007). A Tabu Search Approach to the Strategic Airlift Problem. *Military Operations Research*, 59-79.

Lee, J., Lukachko, S., & Waitz, I. (2004). Aircraft and Energy Use. *Encyclopedia of Energy*, 29-38.

Lei, H., Laporte, G., & Guo, B. (2011). The Capacitated Vehicle Routing Problem with Stochastic Demands and Time Windows. *Computers & Operations Research, 38*(12), 1775-1783.

Lewis, I. (1998). The civil reserve air fleet: Balancing risks and incentives. *Transportation Journal*, 32-39.

Longo, H., de Aragão, M. P., & Uchoa, E. (2006). Solving capacitated arc routing problems using a transformation to the CVRP. *Computers & Operations Research, 33*(6), 1823-1837.

Lund, J. B. (1993). *An Assessment of Strategic Airlift Operational Efficiency.* Project Air Force Analysis of the Air War in the Gulf.

Lurdes, M., Anutnes, B., & Pinelo, A. (1990). Airport pavement evaluation and ACN-PCN classification. *Third international conference on bearing capacity of roads and airfields.* Trondheim, Norway: Tapir Publishers.

Martin, J. C., & Voltes-Dorta, A. (2011). The dilemma between capacity expansions and multi-airport systems: Empirical evidence from the industry's cost function. *Transportation Research Part E: Logistics and Transportation Review*, 382-389.

Mazraati, M. (2010). *World Aviation Fuel Demand Outlook.* Metropolitan University: OPEC Energy Review.

Mihram, G., & Nolan, R. (1969). A Stochastic Simulation of the Strategic Airlift System. *Proceedings of the Third Conference on Apllications of Simulation*, (pp. 184-192).

Miravite, A., & Schlegel, C. F. (2006). Global Enroute Basing Infrastructure Location Model. *Air Force Institute of Technology Graduate Research Paper*.

Morton, D. P., Rosenthal, R. E., & Weng, L. T. (1995). Optimization Modeling for Airlift Mobility . *(No. NPS-OR-95-007). Naval Postgraduate School Monterey CA Dept of Operations Research*.

Murphy, P., Dalenberg, D., & Daley, J. (1989). Improving international trade efficiency: airport and air cargo concerns. *Transportation Journal*, 27-35.

Nagata, Y., Bräysy, O., & Dullaert, W. (2010). A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research, 37*(4), 724-737.

Naylor, R. (2009). Improving and Extending the Mobility En Route System. *Air Force Institute of Technology Thesis*.

Nielsen, C. A., Armacost, A. P., Barnhart, C., & Kolitz, S. E. (2004). Network Design Formulations for Scheduling US Air Force Channel Route Missions. *Mathematical and Computer Modelling, 39*(6), 925-943.

Oster , C. V., Strong, J. S., & Zorn, C. K. (2013). Analyzing aviation safety: problems, challenges, opportunities. *Research in Transportation Economics*, 148-164.

Owen, M. (2008). *Fuel Efficiency Development and Prediction.* OMEGA, Manchester Metropolitan University.

Pascal, B. (1665). *Traité de Triangle Arithmetique: avec quelques autres petits traiter sur la mesme matiere.* Chez Guillaume Desprez.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research, 34*(8), 2403-2435.

Rappoport, H. K., Levy, L. S., Golden, B. L., & Feshbach, D. S. (1991). Estimating Loads of Aircraft in Planning for the Military Airlift Command. *Interfaces, 21*(4), 63-78.

Rappoport, H. K., Levy, L. S., Golden, B. L., & Toussaint, K. J. (1992). A Planning Heuristic for Military Airlift. *Interfaces, 22*(3), 73-87.

Rathi, A. K., Church, R. L., & Solanki, R. S. (1992). A Macro Level Analysis of the Airlift Deployment Problem. *Computers & Operations Research, 19*(8), 731-742.

Regulations, C. o. (2010). *Title 14: Aeronautics and Space.* Office of the Secretary, Department of Transportation, Sec 19-5.

Reiman, A., Johnson, A., & Cunningham, W. (2011). Competitive Advantage and Fuel Efficiency in Aviation. *Journal of Transportation Management, 22*(2), 75-91.

Reiman, A., Weir, J., Johnson, A., & Dube, T. (2014). Distance Value Model for Nodal Reduction of the Strategic Airlift Problem. *Pending Publication*.

Rink, K. A., Rodin, E. Y., Sundarapandian, V., & Redfern, M. A. (1999). Routing Airlift Aircraft by the Double-Sweep Algorithm. *Mathematical and Computer Modelling, 30*(5), 133-147.

Ruan, Q., Zhang, Z., Miao, L., & Shen, H. (2011). A Hybrid Approach for the Vehicle Routing Problem with Three-Dimensional Loading Constraints. *Computers & Operations Research*.

Rutherford, D., & Zeinali, M. (2009). Efficiency Trends for New Commercial Jet Aircraft. *International Council on Clean Transportation.* Washington DC.

Samm, S., & Perelli, L. (1982). *Estimating Aircrew Fatigue: A technique with Application to Airlift Operations.* Brooks AFB, TX: School of Aerospace Medicine.

Schein, E. (1984). Coming to a New Awareness of Organizational Culture. *Sloan Management Review, 25*(2), 3-15.

Schmenner, R. W. (2001). Looking Ahead by Looking Back: Swift, Even Flow in the History of Manufacturing. *Production and Operations Management, 10*(1), 87-96.

Schmenner, R. W. (2004). Service Businesses and Productivity. *Decision Sciences, 35*(3), 333-347.

Schmenner, R. W., & Swink, M. L. (1998). On Theory in Operations Management. *Journal of Operations Management, 17*(1), 97-113.

Sere, M. (2005). *Strategic Airlift En Route Analysis and Considerations to Support the Global War on Terrorism.* Wright Patterson AFB, OH: Air Force Institute of Technology Graduate School of Engineering Management.

Thomchick, E. (1993). The 1991 Persian Gulf War: short-term impacts on ocean and air transportation. *Transportation Journal*, 40-53.

Tryon, J. E. (2005). An Evaluation of Contingency Construction Methods Using Value Focused Thinking. *Air Force Institute of Technology Thesis*, 1-116.

USAF. (2011). *Air Force Instruction 11-2MDS Volume 3.* Scott AFB, IL: HQ AMC/A3V.

Vincenty, T. (1975). *Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations.* FE Warren AFB, Wyoming: DMAAC Geodetic Survey Squadron.

Watson, F. (2003). *The Air Mobility Planner's Calculator: Improvements, Verification and Validation.* Dayton, OH: Air Force Institute of Technology.

Weir, J. D., & Johnson, E. L. (2004). A three-phase approach to solving the bidline problem. *Annals of Operations Research*, 283-308.

Wilkins, D. E., Smith, S. F., Kramer, L. A., Lee, T. J., & Rauenbusch, T. W. (2008). Airlift Mission Monitoring and Dynamic Rescheduling. *Engineering Applications of Artificial Intelligence, 21*(2), 141-155.

Wu, T. T., Powell, W. B., & Whisman, A. (2009). The Optimizing-Simulator: An Illustration Using the Military Airlift Problem. *ACM Transactions on Modeling and Computer Simulation (TOMACS), 19*(3), 14.

Yamani, A., Hodgson, T., & Martin-Vega, L. (1990). Single Aircraft Mid-Air Refueling using Spherical Distances. *Operations Research*, 792-800.

REPORT DOCUMENTATION PAGE		*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE (DD–MM–YYYY)	2. REPORT TYPE	3. DATES COVERED *(From — To)*
18-09-2014	Dissertation	Sep 2011 - Sep 2014

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Enterprise Analysis of Strategic Airlift to Obtain Competitive Advantage through Fuel Efficiency	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Reiman, Adam D., Lieutenant Colonel, USAF	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN), 2950 Hobson Way WPAFB, OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENS-DS-14-S-16
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) United States Transportation Command Joint Distribution Process Analysis Center Attn: Pat Mcleod 508 Scott Drive DSN: 770-5238 Scott Air Force Base, IL 62225-5357 Patrick.k.mcleod.civ@mail.mil	10. SPONSOR/MONITOR'S ACRONYM(S) USTRANSCOM/TCAC
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Statement A: Approved For Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

The rising cost of fuel has led to increasing emphasis on fuel efficiency in the aviation industry. As fuel costs become a larger proportion of total costs, those entities with a dynamic capability to increase their fuel efficiency will obtain competitive advantage. Assessing cargo throughput and fuel efficiency requires the creation of all routes of potential value for a given set of requirements that need to be airlifted from source to destination airfield. The time required for route computation can be significantly reduced through the use of nodal reduction. Use of the proposed model can assist evaluation of enterprise wide efficiency and effectiveness.

15. SUBJECT TERMS
Strategic Airlift Problem, Fuel Efficiency, Value Model, Nodal Reduction, Cargo Throughput

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jeffery D. Weir, AFIT/ENS
U	U	U	UU	205	19b. TELEPHONE NUMBER (Include Area Code) (937) 255-3636 x0000 Jeffrey.Weir@afit.edu